living with "dirty" data

(while avoiding exascale "garbage in, garbage out")

Mike McKerns

Caltech

ExMatEx

Extreme Materials at Extreme Scale

there will be errors, do we need to care?

- Keynote: "Failure, Resilience, Opportunity and Innovation"
  John Daly, Department of Defense

- How will HPC continue to provide insight into the nation's most important and challenging problems using computers that fail regularly and even give wrong answers? Resilience is not about making all of the errors go away. On the contrary, systems intended to run without errors often fail in the most catastrophic ways. Resilience is about understanding how systems fail and creating applications that can fail their way to success.

- …so, should we really care about failure?
- Is there an algorithm that can tell us whether we should care?
- Can we use this algorithm to redesign our calculations so that we needn't worry about failure?

the challenge: resilience at the exascale

- It's expected failure will occur as a part of normal operation.
  - I'll focus on "bad data", but in many cases I'll generalize to "failure"
- What algorithmic changes can enable resilience at exascale?
  - conjecture: an exascale system should be driven by statistics, and utilize redundancy where failure is expected to have a sizable impact
  - robustness against failure over the need to restart
  - programming models for dynamic flexibility in execution
  - asynchronous parallel and stochastic operation
  - integrated statistical forecasting and metric evaluation
- How can statistics play a huge role in resiliency at exascale?
  - identifying and filtering out errors (e.g. outliers that point to 'bad data')
  - statistical sampling (for known distributions)
  - statistical estimators/validators for system/algorithm behavior
  - integrated driver for algorithmic robustness against failure

resilience and the state machine

- conjecture: if everything, including data, has it's state captured by the system's state machine, any form of failure can be mitigated (or at least recovered from).
  - all state is captured in objects, including data
  - programming models are used to provide flexible dynamic execution

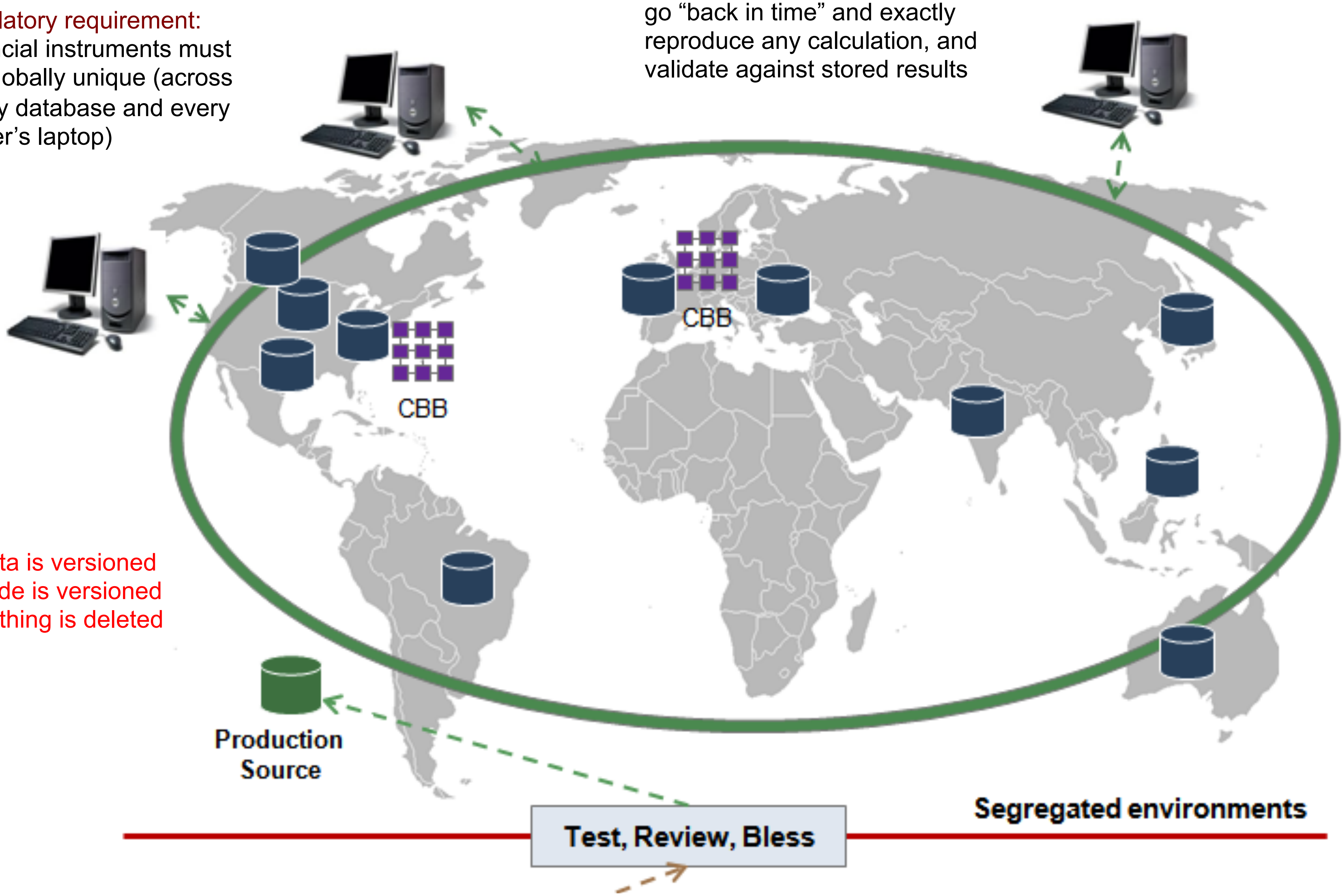
- This design is used by several Tier-1 banks for their global algorithmic trading and market risk systems.
  - all calculations are managed on an abstract syntactic graph
  - all state is captured in objects, and must reside in nodes on the graph
  - the graph itself is an object, and can be stored in a database

- Both the data and the code are treated as objects which are versioned & stored in (memory) instances of NoSQL databases
  - previously, data was stored on disk accessed by M's of processors!

goal: know with certainty what you have

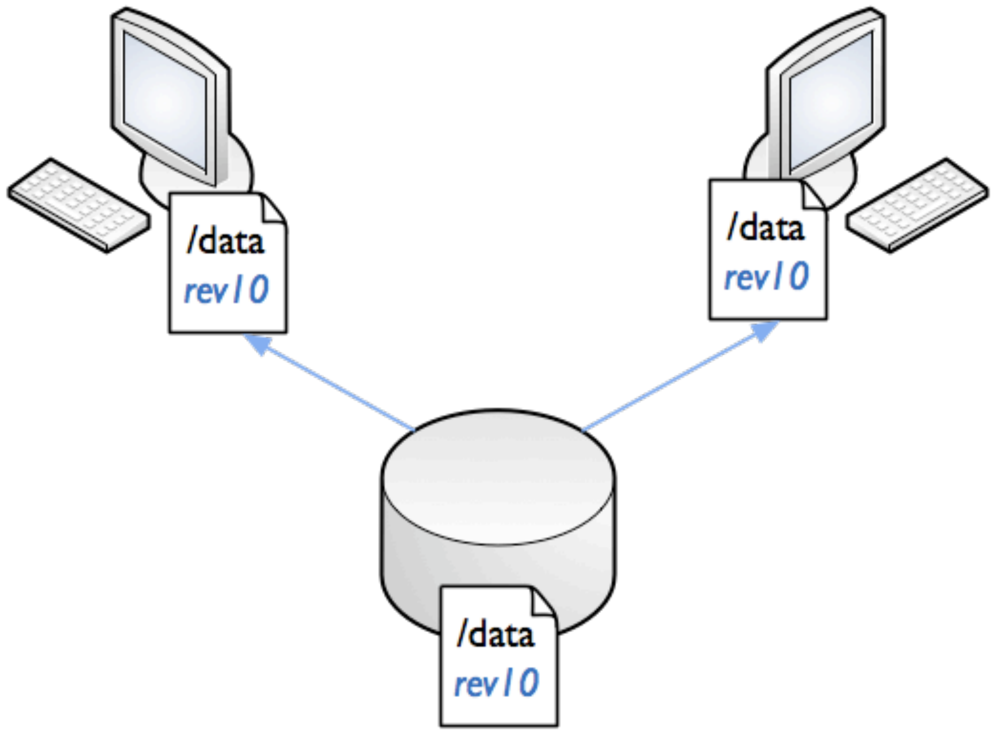
regulatory requirement:
financial instruments must be globally unique (across every database and every trader's laptop)

go "back in time" and exactly reproduce any calculation, and validate against stored results



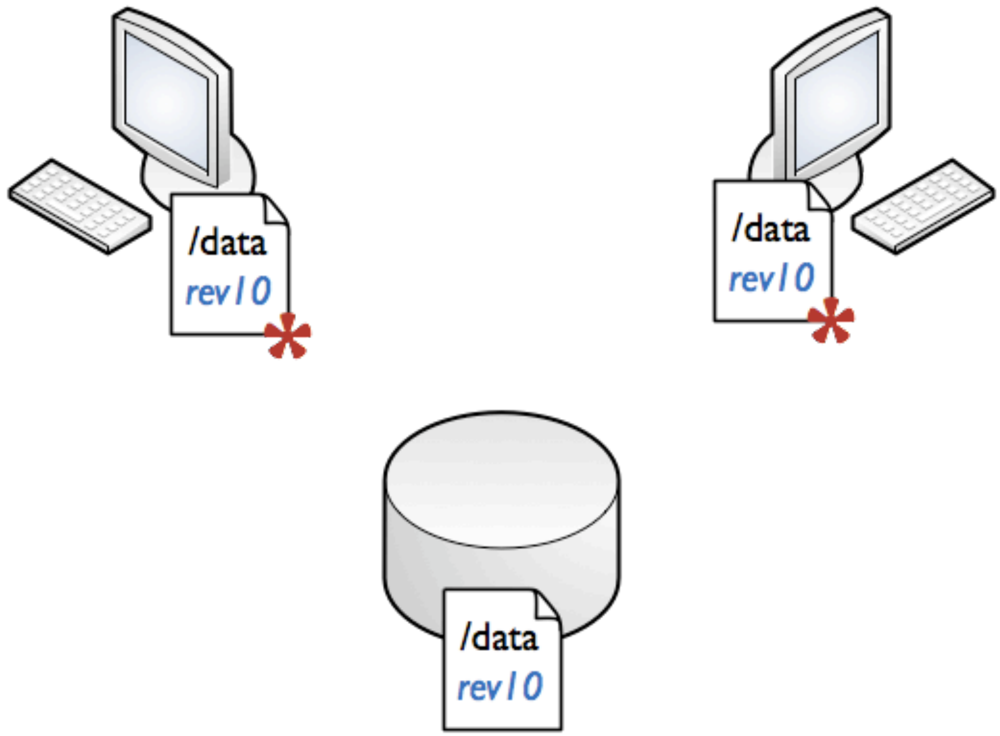
data is versioned
code is versioned
nothing is deleted

data history managed through revisioning



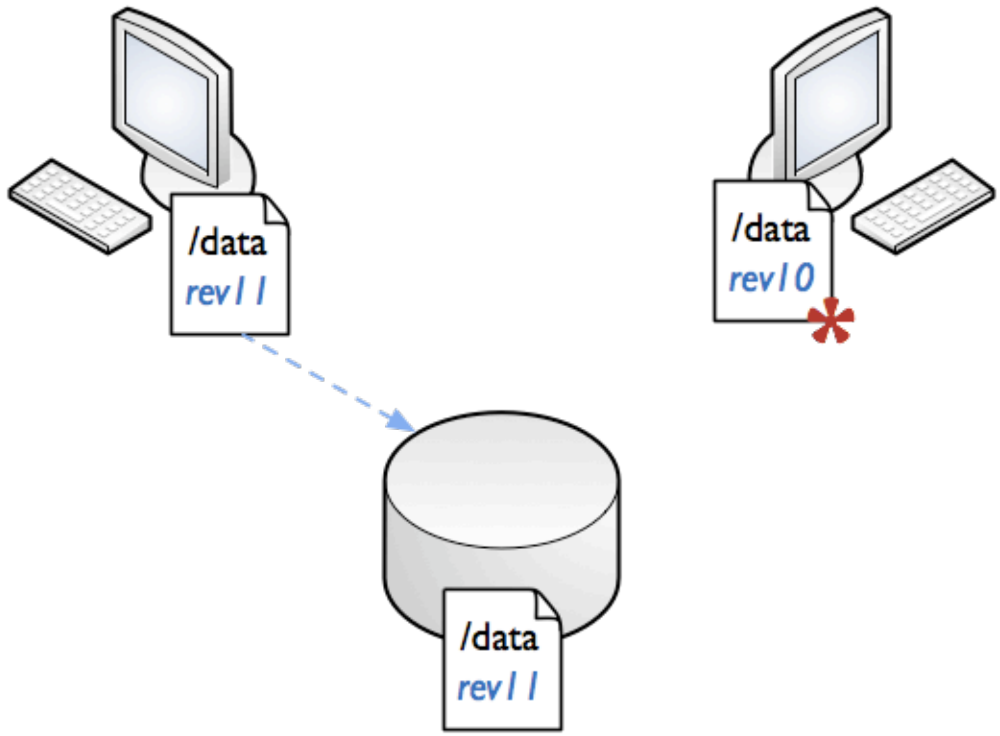
Two clients each get copies of revision 10 of '/data'

data history managed through revisioning



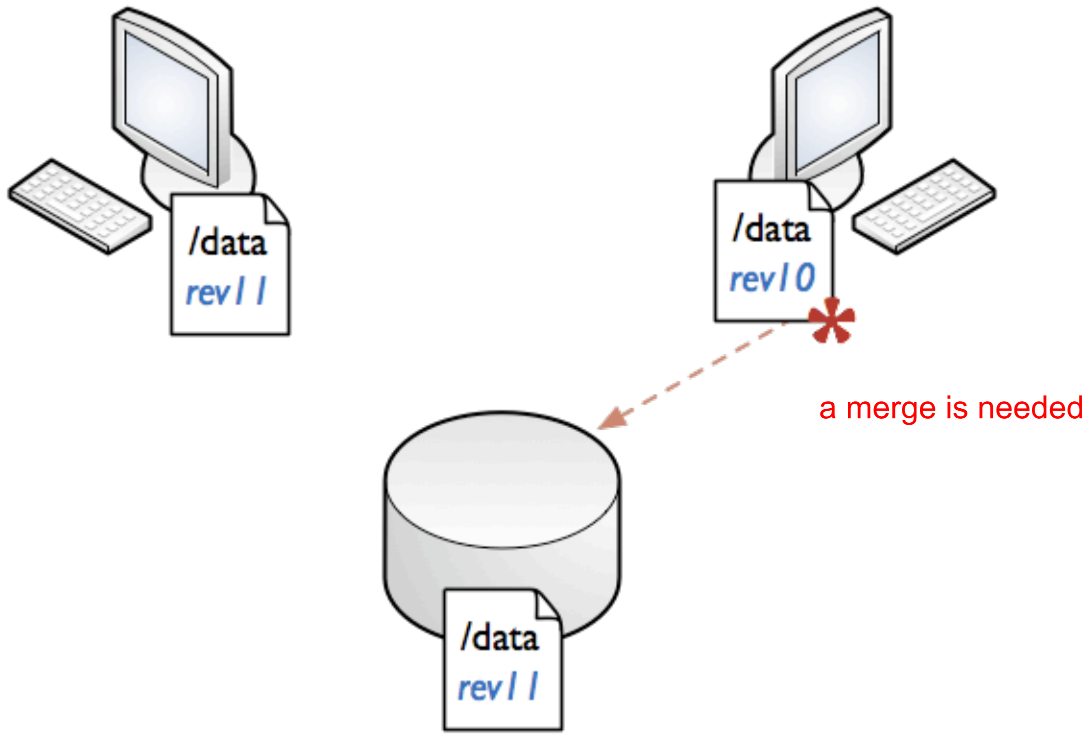
Each client does a calculation that modifies '/data'

data history managed through revisioning



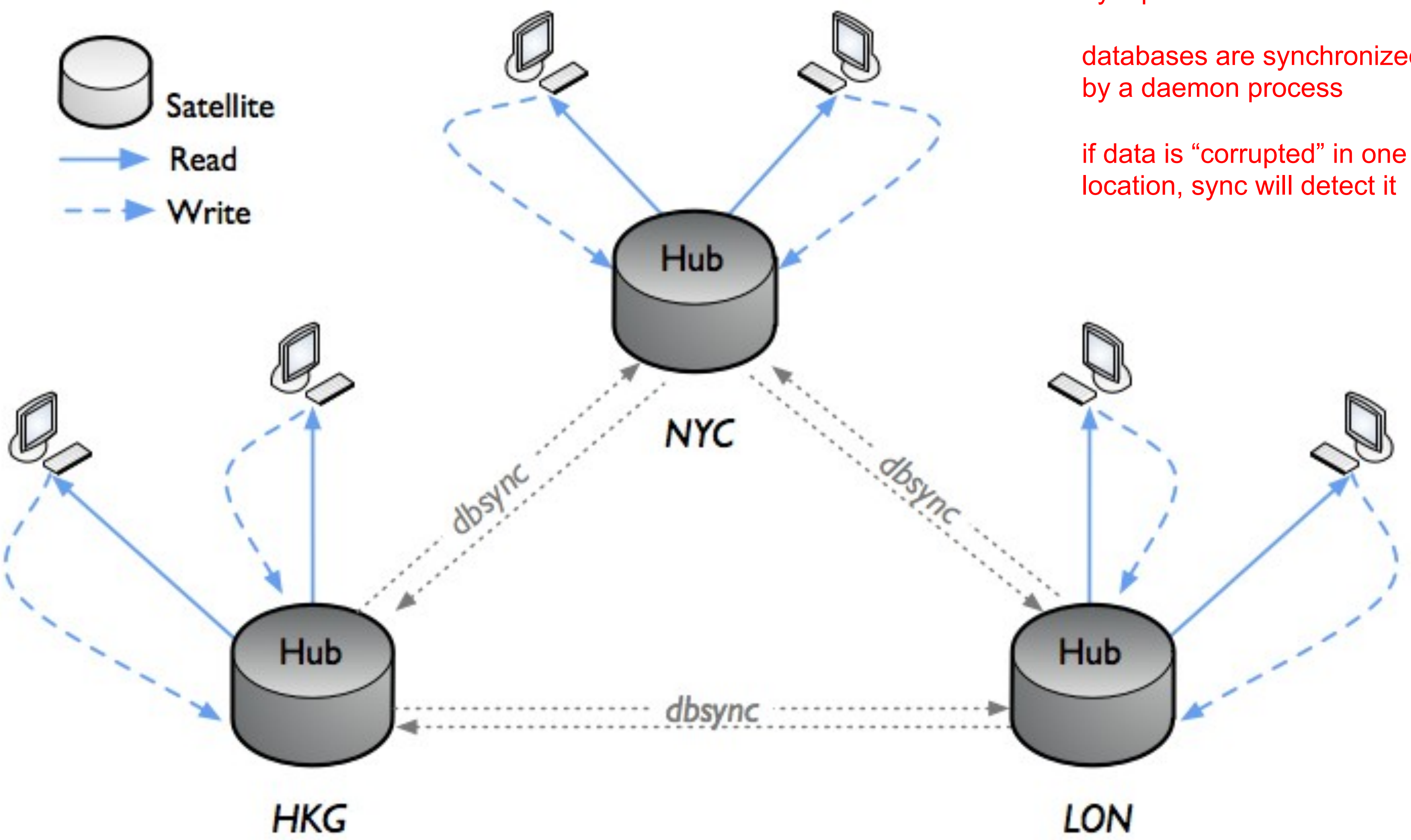
The first client writes '/data' back to the database.
The revision number is increased to 11.

data history managed through revisioning



The second client attempts to write '/data'. The revision numbers don't agree, so the write fails.

global data synchronization is satellite-hub



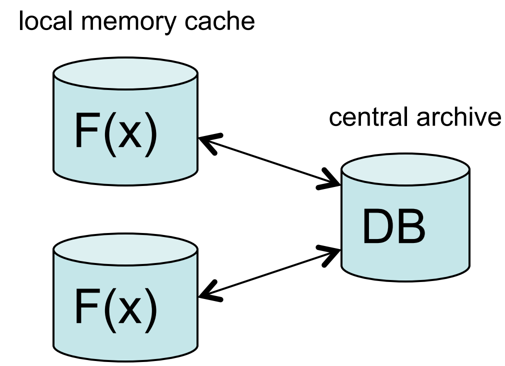
databases can be unioned by a path-order mechanism

databases are synchronized by a daemon process

if data is “corrupted” in one location, sync will detect it

klepto: asynchronous sharing of state

- klepto features:
  - unified API for caching and archiving
  - cache-to-archive interaction strategies
    - lru, lfu, mru, random_replace, …
  - backends: memory, file, memmap, directory, database, db table
  - unified API for key encoding / serialization / hashing / encryption
    - extensible: leverage pickle, json, dill, codecs, md5, … you pick
  - 'ignore' selected arguments (partial arg caching)
  - cache interpolation by rounding
  - can leverage SQLAlchemy, numpy internals

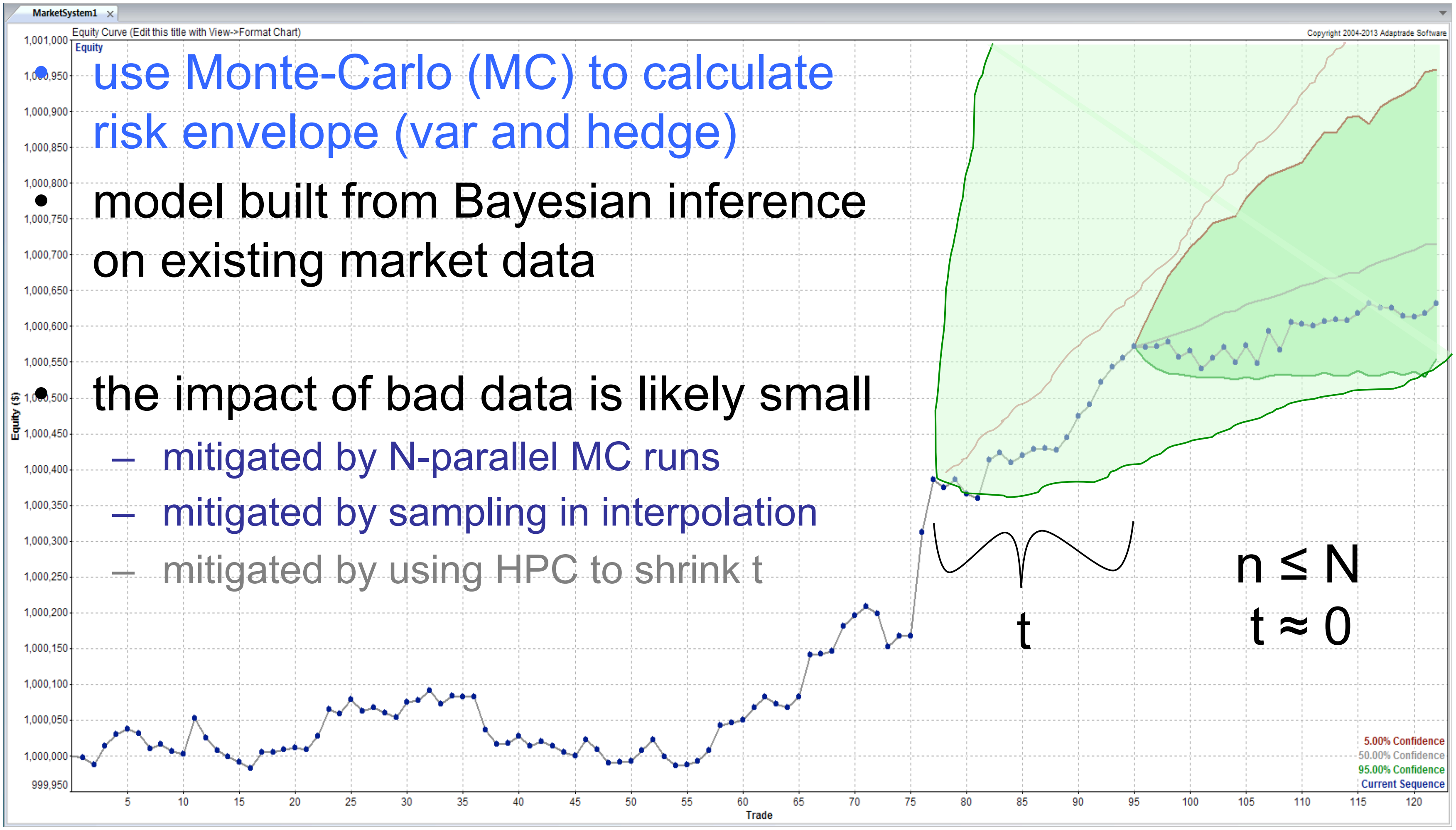


- planned and in-progress:
  - leverage: hdf, redis, shared memory
  - more interpolation algorithms
    - kriging, etc…
  - asynchronous cache-to-archive updates

ExMatEx
Extreme Materials at Extreme Scale

the time-series problem: market risk

- use Monte-Carlo (MC) to calculate risk envelope (var and hedge)
- model built from Bayesian inference on existing market data
- the impact of bad data is likely small
  - mitigated by N-parallel MC runs
  - mitigated by sampling in interpolation
  - mitigated by using HPC to shrink t



- big banks w/ large HPC often perform simple linear statistics
  - speed trumps accuracy

non-accumulating iterative problems

- *time-series* problems map well to data stream analytics
- robust statistics can be applied “in streaming mode”, as results are generated (as opposed to post-mortem)
  - O(N) calculations to produce/process data
  - O(N) calculations required to identify and reject outliers in data
  - calculation of approximate sampling statistics (for known distributions)
  - fast and robust statistics is an area of active research
- failure is generally not catastrophic
  - each time step is non-accumulating
  - the impact of bad data is often contained to a single calculation
  - the more data/updates, the more resilient
- is it a resilience strategy to convert algorithms to this type?
  - asynchronous parallel: speed + resiliency

accumulating iterative problems

- *time-evolution* is not as well suited for data streaming analytics
- robust statistics can be applied “in streaming mode”
  - tend to be larger than O(N)
  - tend to be approximate and fragile
  - also an area of active research
- failure may be catastrophic
  - each time step is accumulating, so errors are generally compounded
  - the impact of bad data is rarely contained to a single calculation
  - may be mitigated by adding redundancy and randomness
  - may be mitigated by validation against expected model error
- materials modeling is generally a time-evolution problem
  - does that mean we cannot convert to robust asynchronous parallel?

time-evolution: a leading question

…

- global optimization underlies almost every flavor of UQ, however is arguably one of the most limiting factors in predictive science – primarily because optimization algorithms are iterative (i.e. “serial”).

…

- can we rethink optimization (and statistics/UQ) to be embarrassingly parallel?

  or maybe better…

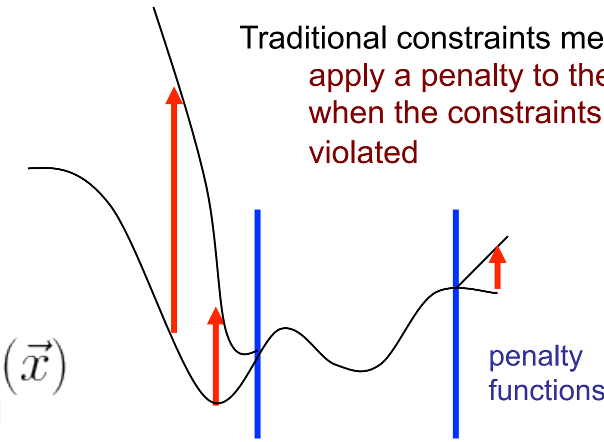
- if you had a global optimizer and exascale computing resources, would you pose statistics/UQ questions differently?

- I have built an optimization framework that is designed to address large-dimensional and highly-constrained non-convex global optimization and rare-event UQ problems. A key aspect of how it works is that an optimizer can dynamically spawn a hierarchy of optimizers to address portions of the problem, and those nested optimizers can also do the same, and so on.
- One caveat is that each nested optimization must not fail to find it’s target.

mystic: scalable constraints operators

```
from mystic.math.measures import mean, spread
from mystic.constraints import with_penalty, with_mean
from mystic.constraints import quadratic_equality

# build a penalty function
@with_penalty(quadratic_equality, kwds={'target':5.0})
def penalty(x, target):
  return mean(x) - target

# define an objective
def cost(x):
  return abs(sum(x) - 5.0)

# solve using a penalty
from mystic.solvers import fmin
x = array([1,2,3,4,5])
y = fmin(cost, x, penalty=penalty)
```

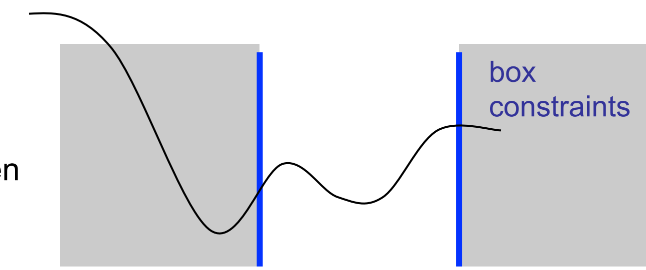
Traditional constraints methods
apply a penalty to the cost when the constraints are violated



$$\phi(\vec{x}) = f(\vec{x}) + k \cdot p(\vec{x})$$

fast, but implicit, inaccurate, and can add spurious features

Decoupling constraints often creates a central convex optimization



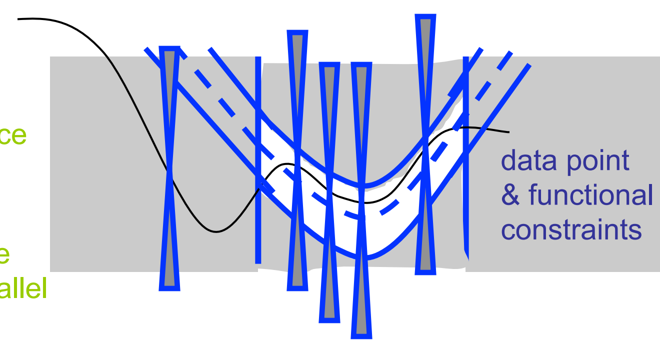

```
# build a functional constraint
@with_mean(5.0)
def constraint(x):
  return x

# solve using constraints
y = fmin(cost, x, constraint=constraint)
```

$$\phi(\vec{x}) = f(c(\vec{x}))$$

explicit and can be parallelized, can strongly reduce search space

$$|\Psi' > = \hat{c}|\Psi >$$

operators that commute can be spawned in parallel



pathos: programming model abstractions

PATHOS
a framework for parallel graph management and execution in heterogeneous computing

```
# select and configure a basic monitor
from pathos import Monitor
evalmon = Monitor()

# apply to a user-provided function
@monitored(evalmon)
def identify(x)
  return x

# select and configure a parallel map
from pathos.maps import SlurmMpirunPool
mpi_map = SlurmMpirunPool(8)

# evaluate the model in parallel
y = mpi_map(identify, range(16))
```

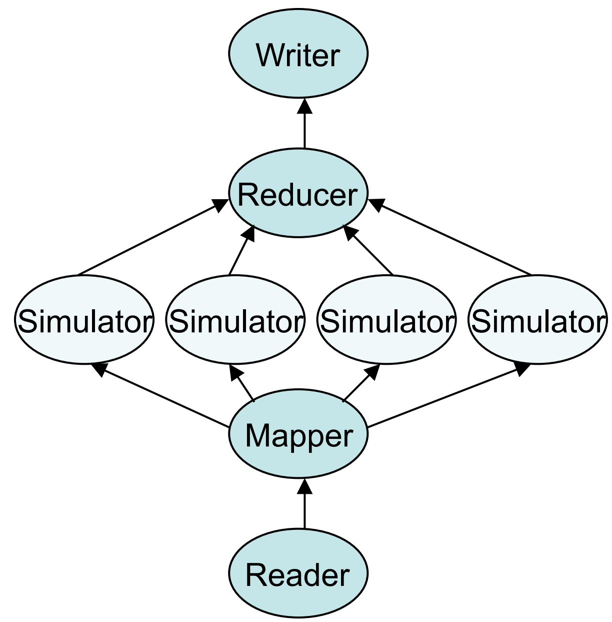
map provides batch processing on an potentially distributed or parallel service

---

```
# select and configure a parallel map
from pathos.maps import IpcPool
ipc_map = IpcPool(2, servers=['foo.caltech.edu'])

# evaluate the model in parallel
y = ipc_map(identify, range(16))
```

typically 80-90% as efficient as hand-tuned parallel code

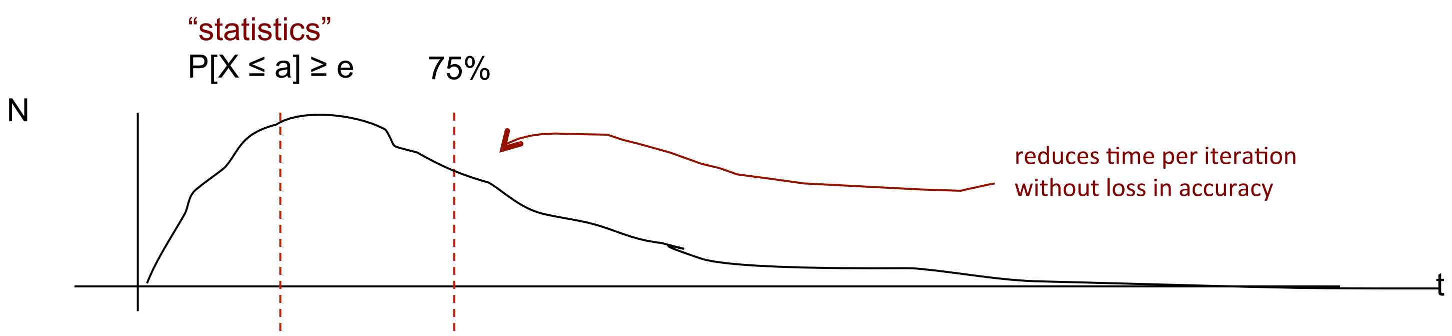
- rapid exploration of system design:
  - communication patterns
  - parallelism hierarchies
  - memory hierarchies
  - synchronization and scheduling
  - resilience strategies
  - system efficiency



asynchronous map: speed and robustness

- blocking map is fragile and prone to failure
- …so decouple the launch and termination of parallel map

- utilize a stop condition for when results are "good enough"
  - simple case: 75% of the results have returned
  - better case: use statistics to determine if "good enough"



- note: we can still collect and archive all launched runs
  - blocking time to the next iteration can be greatly reduced
  - a "condition" removes requirement all runs complete

ExMatEx
Extreme Materials at Extreme Scale

scalability with asynchronous parallelism

- leverage asynchronous parallel computing in optimization
  - optimizers have and can save state (to file or database archive), have streaming diagnostic monitors
  - optimizers are serializable and asynchronous (thus are non-blocking parallel distributed)
  - has slots for parallel maps on the objective, constraints, iteration, and the solver itself (for parallel ensemble and nested solvers)
  - has memory caching and transparent archiving
  - dynamic optimization strategies, compound termination conditions, speed-up with dimensional collapse
  - optimizers are event-based, can react to changing constraints & objective

- constraints operators enable scalable nonlinear optimization
  - apply constraints as an "operator"
  - almost embarrassingly parallel
  - constraints solvers are dynamically launched by a governing optimizer
  - has been used to solve problems with 1000's of nonlinear constraints

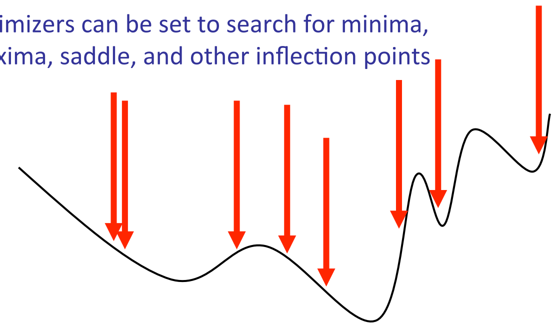
$\lvert\Psi' > = \hat{c}\lvert\Psi >$	$\phi(\vec{x}) = f(c(\vec{x}))$
operators that commute can be spawned in parallel	explicit and can be parallelized, can strongly reduce search space

mystic: massively-parallel optimizers

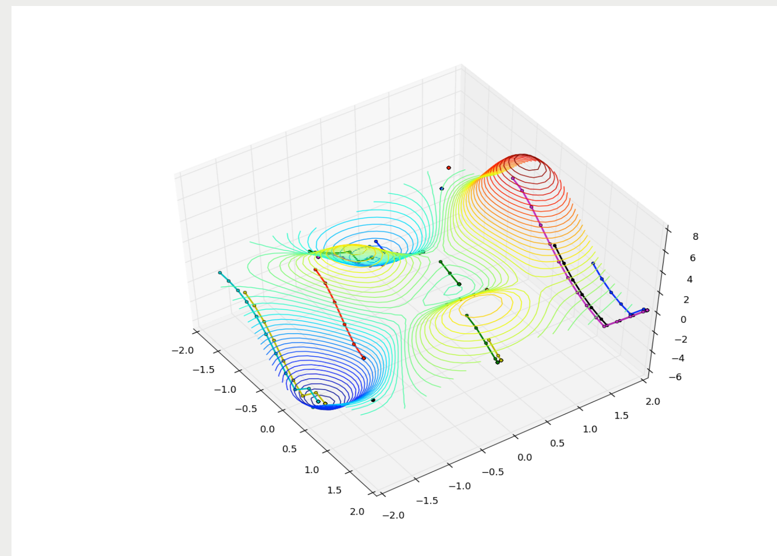
mystic
a framework for highly-constrained non-convex optimization and uncertainty quantification

```
# the function to be minimized and the bounds
from mystic.models import rosen as my_model
lb = [0.0, 0.0, 0.0]; ub = [2.0, 2.0, 2.0]

# get monitor and termination condition objects
from mystic.monitors import LoggingMonitor
stepmon = LoggingMonitor(1, 'log.txt')
from mystic.termination import ChangeOverGeneration
COG = ChangeOverGeneration()

# select the parallel launch configuration
from pyina.launchers import TorqueMpi
my_map = TorqueMpi('25:ppn=8').map

# instantiate and configure the nested solver
from mystic.solvers import PowellDirectionalSolver
my_solver = PowellDirectionalSolver(len(lb))
my_solver.SetStrictRanges(lb, ub)
my_solver.SetEvaluationLimits(1000)

# instantiate and configure the outer solver
from mystic.solvers import BuckshotSolver
solver = BuckshotSolver(len(lb), 200)
solver.SetRandomInitialPoints(lb, ub)
solver.SetGenerationMonitor(stepmon)
solver.SetNestedSolver(my_solver)
solver.SetSolverMap(my_map)
solver.Solve(my_model, COG)
# obtain the solution
solution = solver.bestSolution
```

optimizers can be set to search for minima, maxima, saddle, and other inflection points



with enough optimizers, we get a global map of the potential surface in a single shot



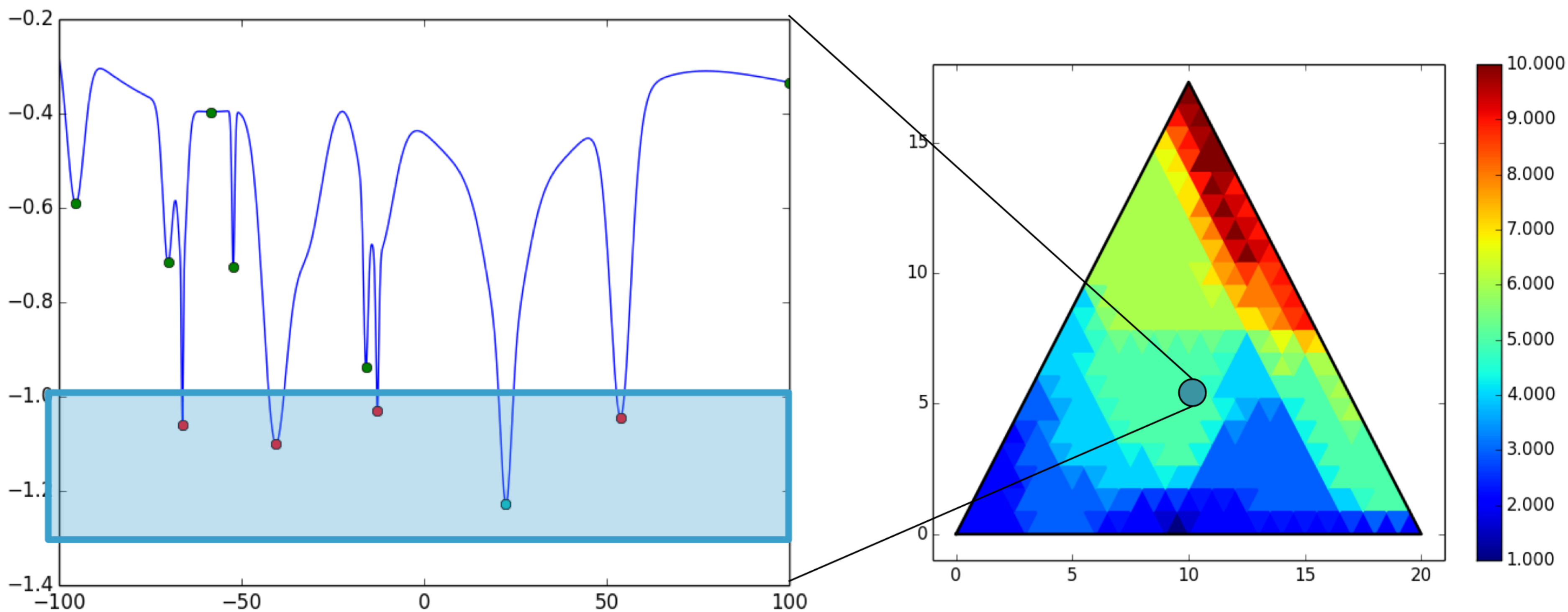
– operators can spawn nested optimizations to solve constraints

example: degeneracy in structure solution

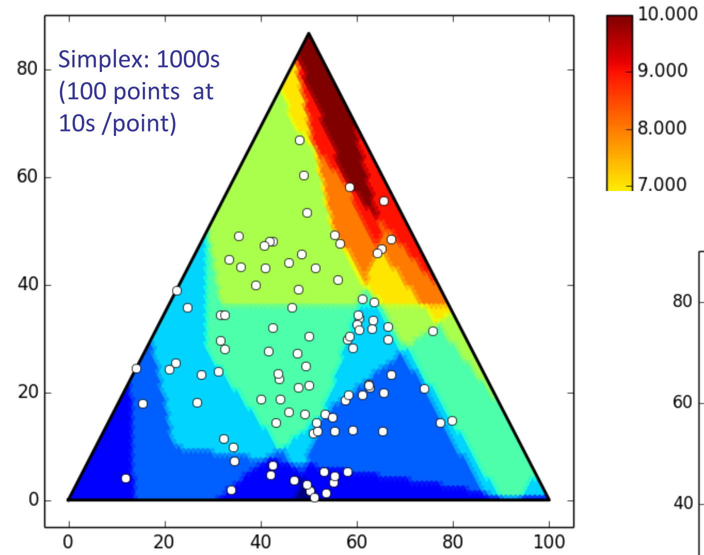
- Solving a 1D proxy problem with multiple degenerate minima, finding the number of such minima.
- Constructed from 3 sets of Gaussians which may be mixed with different weights.

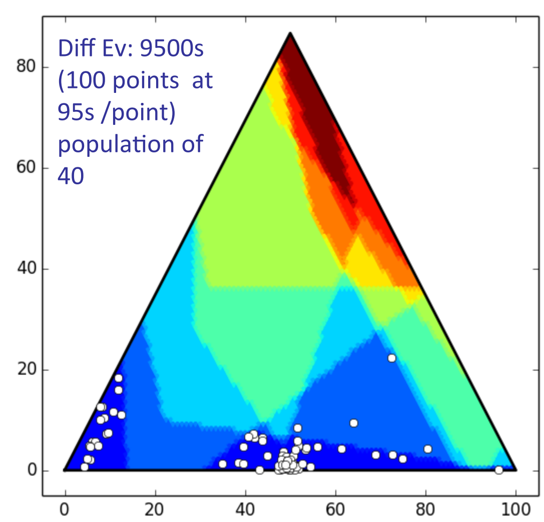
– Step 1: pick a point on the ternary source diagram
– Step 2: find the degeneracy for that version of the target function
– Step 3: use downhill method to choose a new point. Goal is to find point of lowest degeneracy
– Step 4: repeat many times, try different highly parallel searches

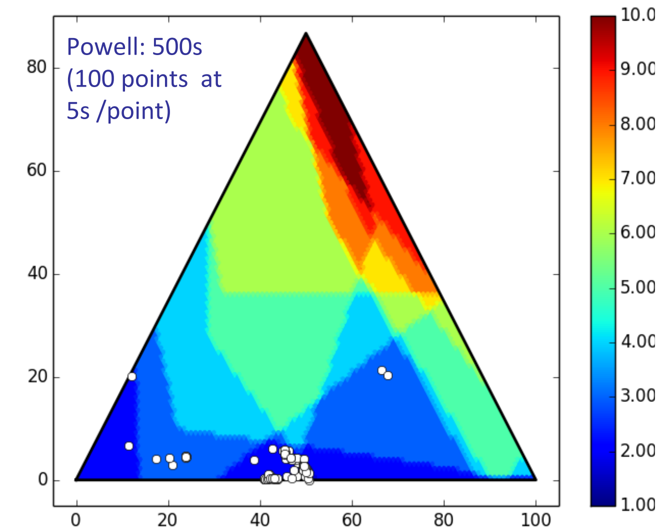
- Problem is hard for solvers because there are large flat regions of the surface.

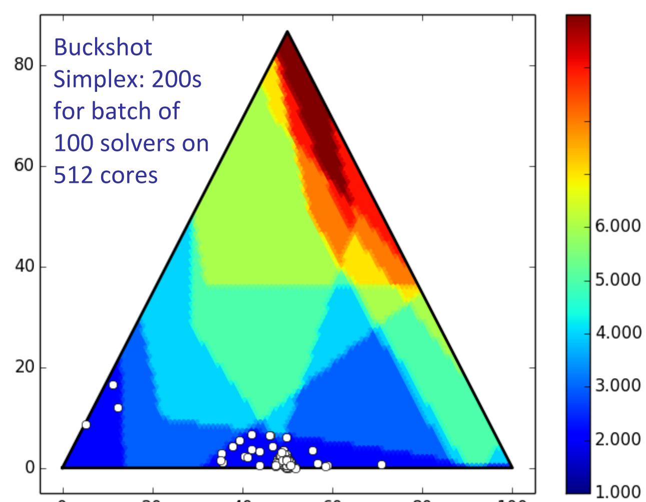


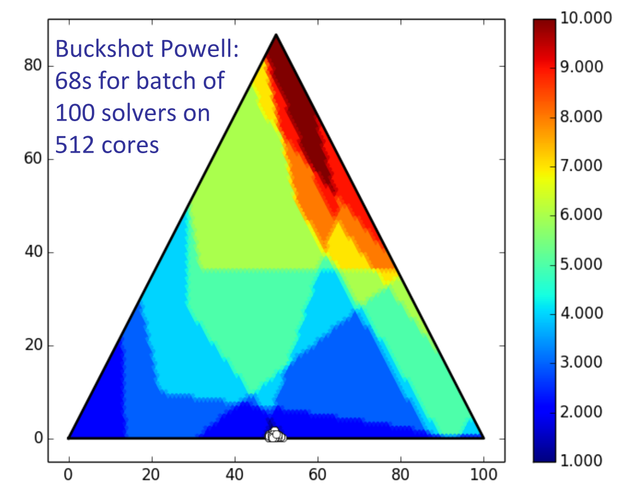
benchmark with ensemble solvers



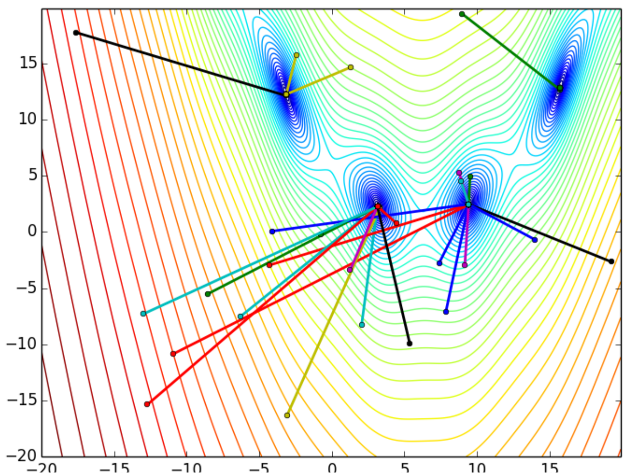






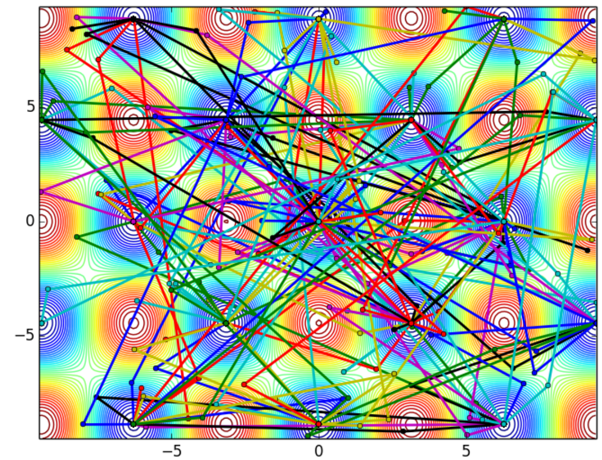


example: ensemble global search

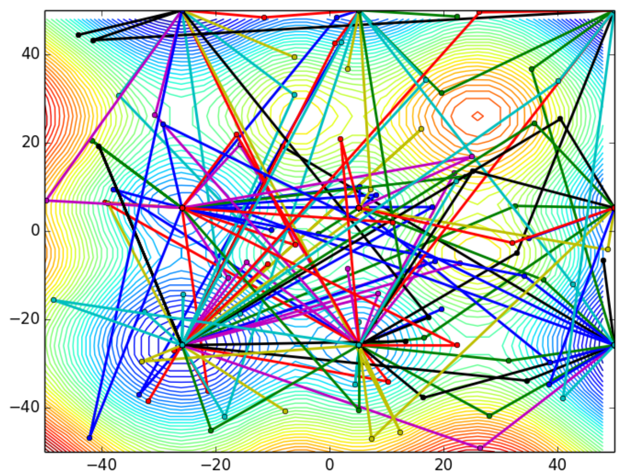


Single Buckshot Powell search for all minima

Six-iteration Buckshot Powell search for all minima.

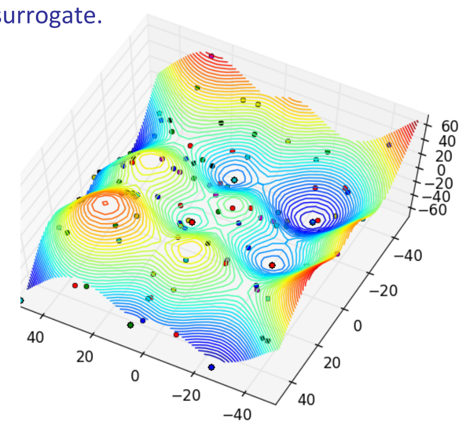



```
dude@hilbert>$ python global_search.py
CacheInfo(hit=12, miss=13, load=0, maxsize=None, size=13)
CacheInfo(hit=18, miss=7, load=0, maxsize=None, size=20)
CacheInfo(hit=22, miss=3, load=0, maxsize=None, size=23)
CacheInfo(hit=24, miss=1, load=0, maxsize=None, size=24)
CacheInfo(hit=25, miss=0, load=0, maxsize=None, size=24)
CacheInfo(hit=25, miss=0, load=0, maxsize=None, size=24)
min: 0.0 (count=1)
pts: 17 (values=6, size=24)
```



Two-iteration Buckshot Powell search for all minima.

Interpolate points to build a surrogate.



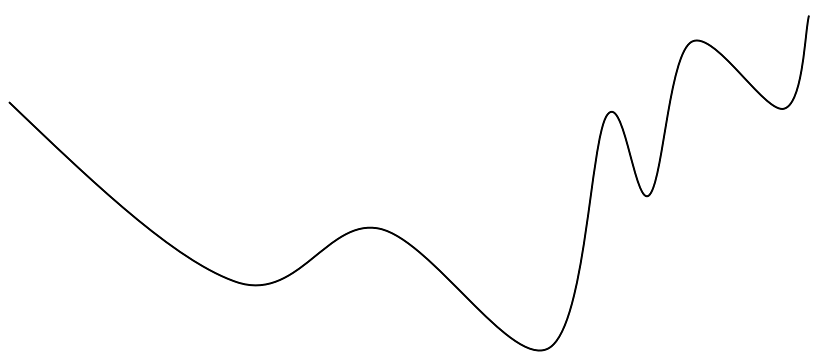

```
dude@hilbert>$ python global_search.py
CacheInfo(hit=17, miss=8, load=0, maxsize=None, size=8)
CacheInfo(hit=24, miss=1, load=0, maxsize=None, size=9)
CacheInfo(hit=25, miss=0, load=0, maxsize=None, size=9)
CacheInfo(hit=25, miss=0, load=0, maxsize=None, size=9)
min: -70.8861291838 (count=1)
pts: 9 (values=8, size=9)
```

“cache” in this case is an abstraction on storage. “load” is local memory cache, while “hit” is an archive hit. “miss” is a new point. Results shown are for when configured for direct connectivity with archival database.

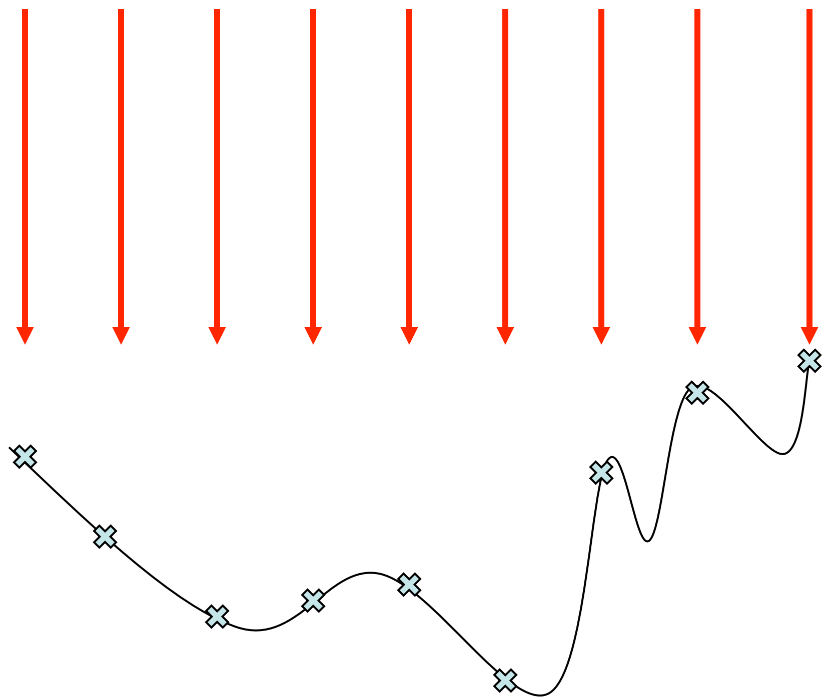
example: building the optimal surrogate

Can we build a surrogate for a n-dimensional surface, where we can optimally replicate the original function's behavior?

You can be smart about it, or use brute force. Let's use brute force.



example: building the optimal surrogate



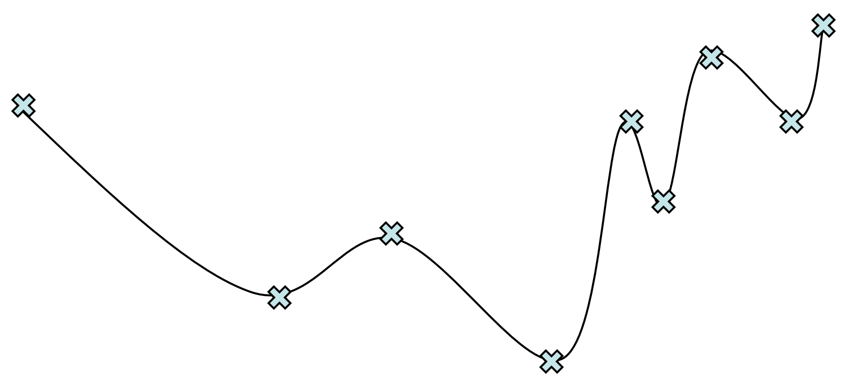
Standard solution: pick a grid density, and drop points on the grid. Then interpolate.

Can we do better?

example: building the optimal surrogate

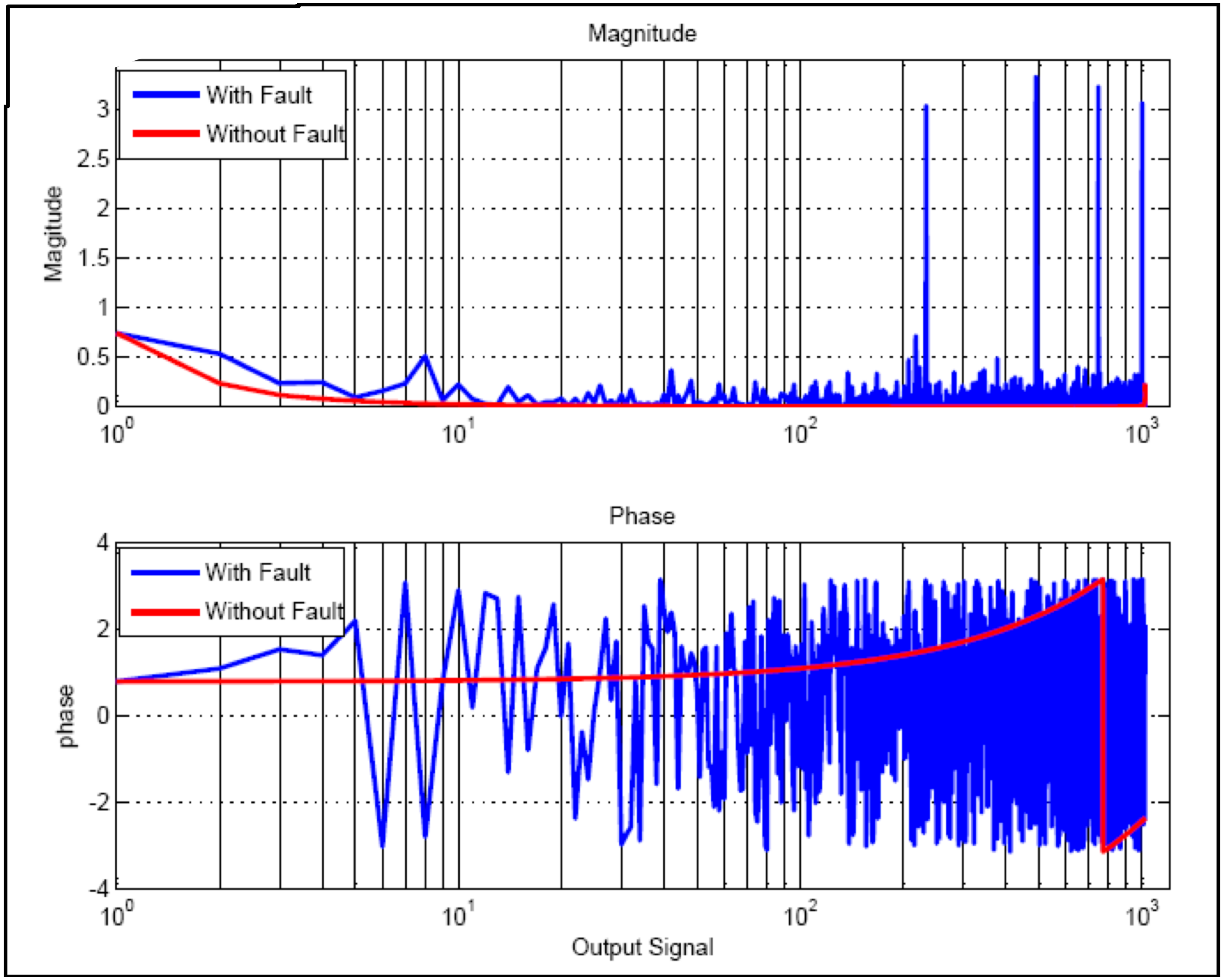
Better solution: pick points at all of the critical points for the unknown surface. Then interpolate.

Need to use an optimizer capable of reliably finding all critical points. Luckily, we have one.



Turning points not shown.
As a "bonus" you also get the points from each function evaluation in the optimization.

can catastrophic failure be a good thing?



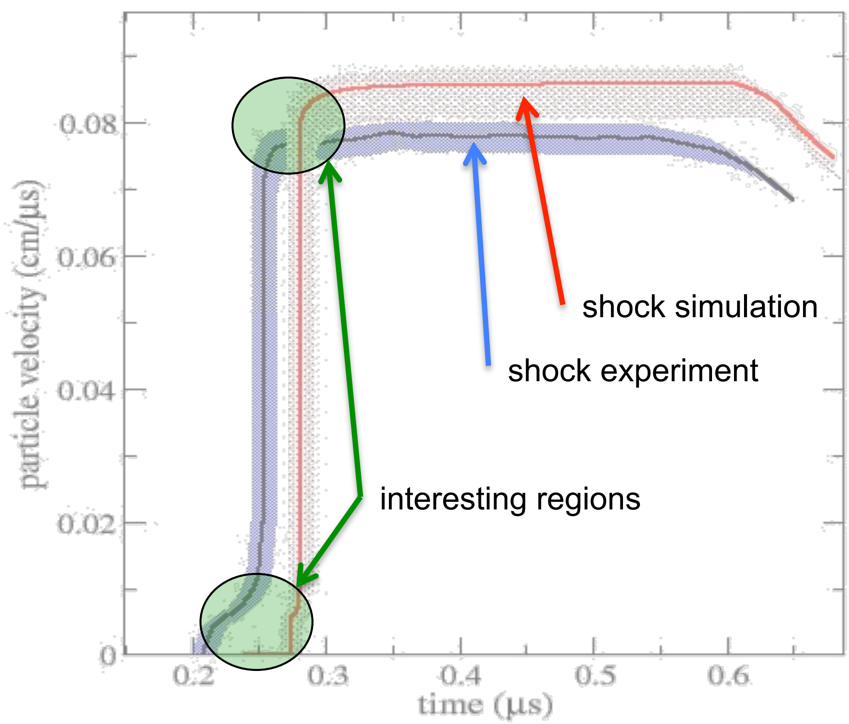
one flipped bit in a FFT can have catastrophic effects

however, catastrophic failure should be easy to detect by examining model error (statistics)

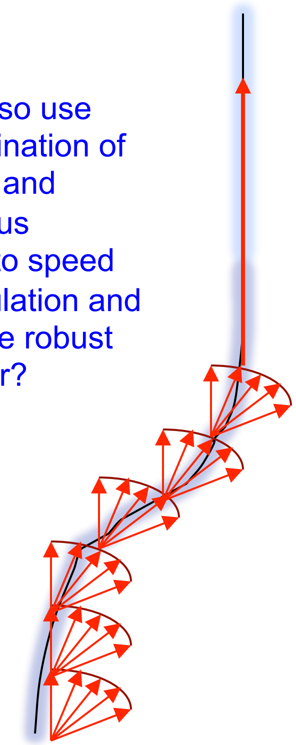
can we leverage model error to provide resilience?

– N. DeBardeleben LANL LA-UR-12-20261

model error in guided shock simulation



Could we also use some combination of model error and asynchronous parallelism to speed up the calculation and make it more robust against error?



In shock simulations, we typically construct a model, then after we simulate we try to determine "misfit" (i.e. model error). We then readjust parameters, and try again.

What if we had a process to build a shock model that was guaranteed to satisfy model error constraints everywhere requested?

What if, as a bonus, the process was resilient to catastrophic failure?

Ahead of each "model-error guided" model evaluation, we run a burst of "model-error guided" surrogate model evaluations. We try to forecast the next coarse model evaluation point. Also, if the surrogate performs well, switch to the surrogate.

Could something like this work?

statistics will play a huge role at exascale

- exascale systems should have to operate under failure
- individual components of the system and individual bits of data should not be trusted… however, the entire system and the data should be trusted **with statistical confidence**.
  - we have to build algorithms that are robust with statistical confidence
- in certain cases, we can measure performance with an estimator (e.g. a model that produces a projected value):
  - failure that is governed by a normal (or at least a known) distribution
  - failure that is not catastrophic (i.e. errors do not compound)
  - when we have built and validated a statistical estimator for the system
  - when we can't do any better
- otherwise we need to determine best and worst case bounds as well as the average case
  - the bounds and the average provide a true system performance measure

probability theory versus uncertainty

- subtle: approximations make the problem "solvable"
  - however, often remove the ability to predict high-impact rare events
- problem typically reduced to one of probability theory
  - classic probability theory by Laplace published in 1812
  - modern probability theory by Kolmogorov published in 1933
  - probability distributions are approximated as a KNOWN
  - standard deviations are used to "reintroduce" the UNKNOWN
- how differ from rigorous calculations of risk and uncertainty?
  - probability distributions are an UNKNOWN
  - unified uncertainty theory by Owhadi published in 2013
- example: picking a red ball from a bag of 100 colored balls
  - *probability*: if 10 balls are red, what's the likelihood in picking a red ball?
  - *uncertainty*: if **on average** 10 balls are red, what's the likelihood of picking a red ball the next time? What's the worst case and best case?

why is catastrophic failure hard to predict?

- hardly anyone solves the "full problem"
  - problems are high-dimensional, nonlinear, and non-convex
  - real-world problems are usually considered "too big" to solve: too many parameters, too complex, etc…
- composing reduced problems with valid strong approximations is an area of active research
  - calculations are expensive and require parallel computing
  - the majority of the effort is often in finding a "best" model or probability distribution or prior
  - once a "best" model/distribution is found, prediction and estimation are separate and often quick calculations
  - iterative and renormalization steps can be used when predictions are found to conflict with problem constraints
- typical: use a prior and fix a probability distribution
  - sampling off a fixed distribution can only predict rare events that have been observed (to inform the prior)
  - can predict average behavior (given enough data), however fails to predict high-impact rare events

- standard approximations:
  - convexity
  - if the objective is expensive, use a less expensive (approximate) surrogate
  - if data exists, use a best-fit surrogate to represent the data (throwing away data)
  - worst: we extract a probability distribution from the data, assuming all future data matches the existing distribution

Bayesian inference, machine learning, MCMC and other standard techniques all use this approach.

example: seismic safety assessment

- Problem: Can we certify the seismic safety of a given structure subjected to earthquake ground motion, where only the maximum magnitude and focal distance of the earthquake are known?

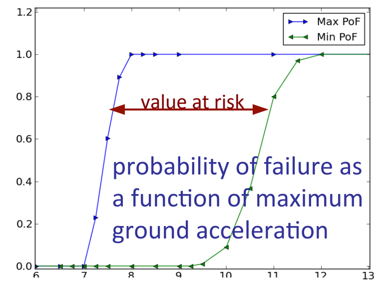
- We construct all possible earthquake scenarios
  - Random inputs of high-dimensionality (~600) with a large number of constraints (~1200)
  - Inputs are coefficients $\boldsymbol{c_i}$ in the transfer function, and amplitudes $\boldsymbol{X_i}$ and durations $\boldsymbol{s_i}$ in the earthquake source function

$$s(t) := \sum_{i=1}^{B} X_i\, s_i(t) \qquad \psi(t) := \frac{\sqrt{q}}{\tau'} \sum_{i=1}^{q} c_i\, \varphi_i(t)$$

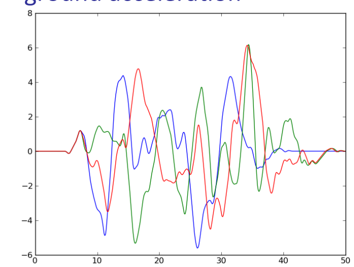
- Ground acceleration is a convolution of the source and transfer functions, while dynamics of joint deflection are governed by

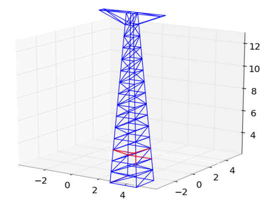
$$v_\alpha(t) = -\int_0^t \mathrm{e}^{-\zeta_\alpha \omega_\alpha (t-\tau)} \sin[\omega_\alpha (t-\tau)] \big(q_\alpha^T M T \ddot{u}_0(\tau)\big) \frac{\mathrm{d}\tau}{\omega_\alpha}$$

$$\ddot{u}_0(t) := (\psi \star s)(t)$$

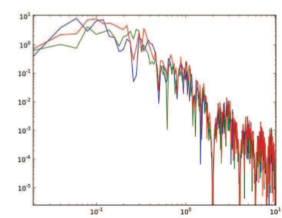


typical scenarios for resulting ground acceleration





a truss structure



when axial strain occurs near truss resonance modes, failure can occur

- Failure occurs when axial strain in any truss member exceeds the member yield strain

$$\|L_i v\|_\infty < S_i$$

- We determine the probability of non-elastic failure with respect to the unknown earthquake ground motion the structure will experience

Owhadi et al, SIAM Review 2012

assumptions have consequences

- An admissible set of scenarios can be constructed by considering the mean power spectrum

$$\mathcal{A}_{\mathsf{MA}} := \left\{ \mu \;\middle|\; \begin{array}{c} \mu \text{ is a prob. dist. on ground motions,} \\ \text{and } \mathbb{E}_\mu[\text{power spectrum}] = s_{\mathsf{MA}} \end{array} \right\} \qquad s_{\mathsf{MA}}(\omega) := C_1 e^{C_2 M_{\mathrm{L}}} \frac{\omega_{\mathrm{g}}^2 \omega^2}{(\omega_{\mathrm{g}}^2 - \omega^2)^2 + 4\xi_{\mathrm{g}}^2 \omega_{\mathrm{g}}^2 \omega^2}$$

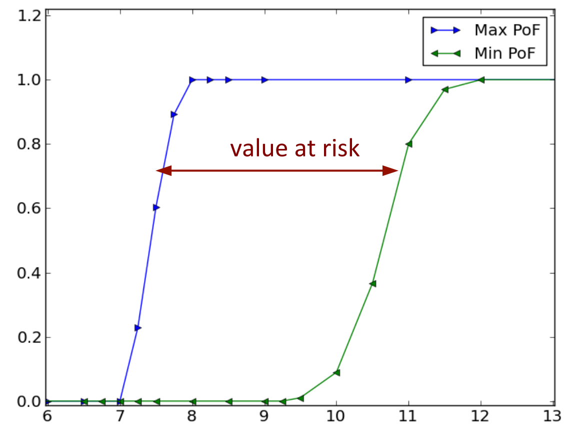
- The typical approach is to repeatedly sample white noise, then filter the samples through a given shape function to generate samples with a "typical" power spectrum
  - amounts to a sampling from only one of the possible probability distributions
  - results are dependent on how well the selected probability distribution applies to all possible scenarios (e.g. are outliers important?)

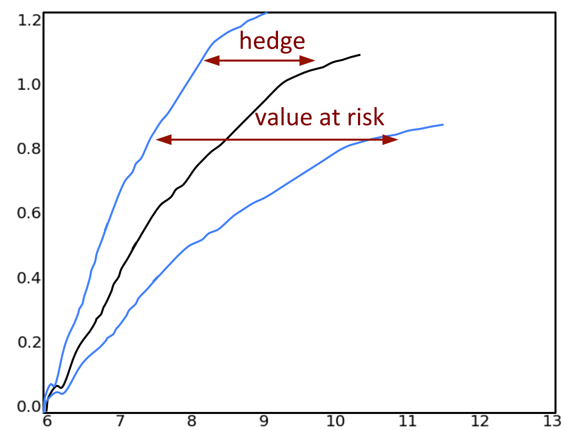
- This approach builds the "best" model based on past events, and hopes futures can be predicted explicitly from the past.

the problem is…

- **The past is not generally a good predictor of the future**

- In general, we have two problem types:
  - "best" case is easy to approximate
    - seismic safety
    - casualty estimates
  - "average" case is easy to approximate
    - stock market futures
    - weather forecasting
    - algorithmic performance





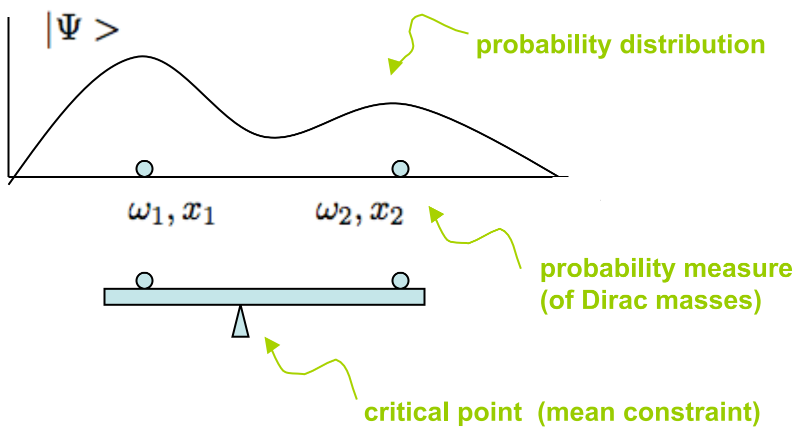
- Finding the remaining information is sketchy
  - bounds found by standard deviations
  - bounds found with monte carlo simulations
  - bounds cannot be approximated

Poor approximations built into our statistical methods often lead to increasing confidence in incorrect results

UQ with unknown probability distributions

- min/max on probability measure space (not input parameter space)

$$|\Psi' > = \hat{c}|\Psi > = \sum_i \omega_i |x_i >$$

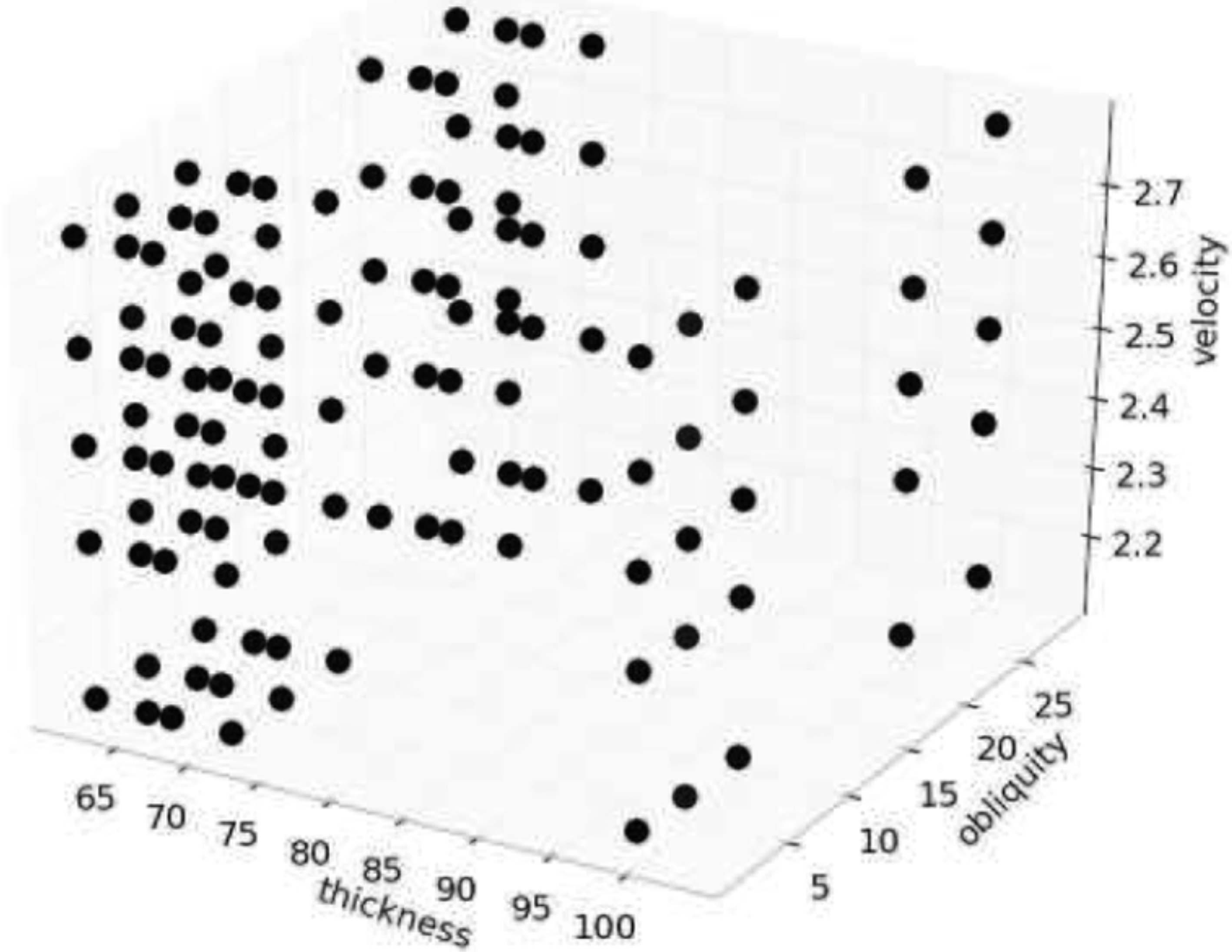


- mean-constrained optimization balances weights and positions of Dirac masses around a critical point

how many points are required? N+1 or less, where N is the number or constraints.

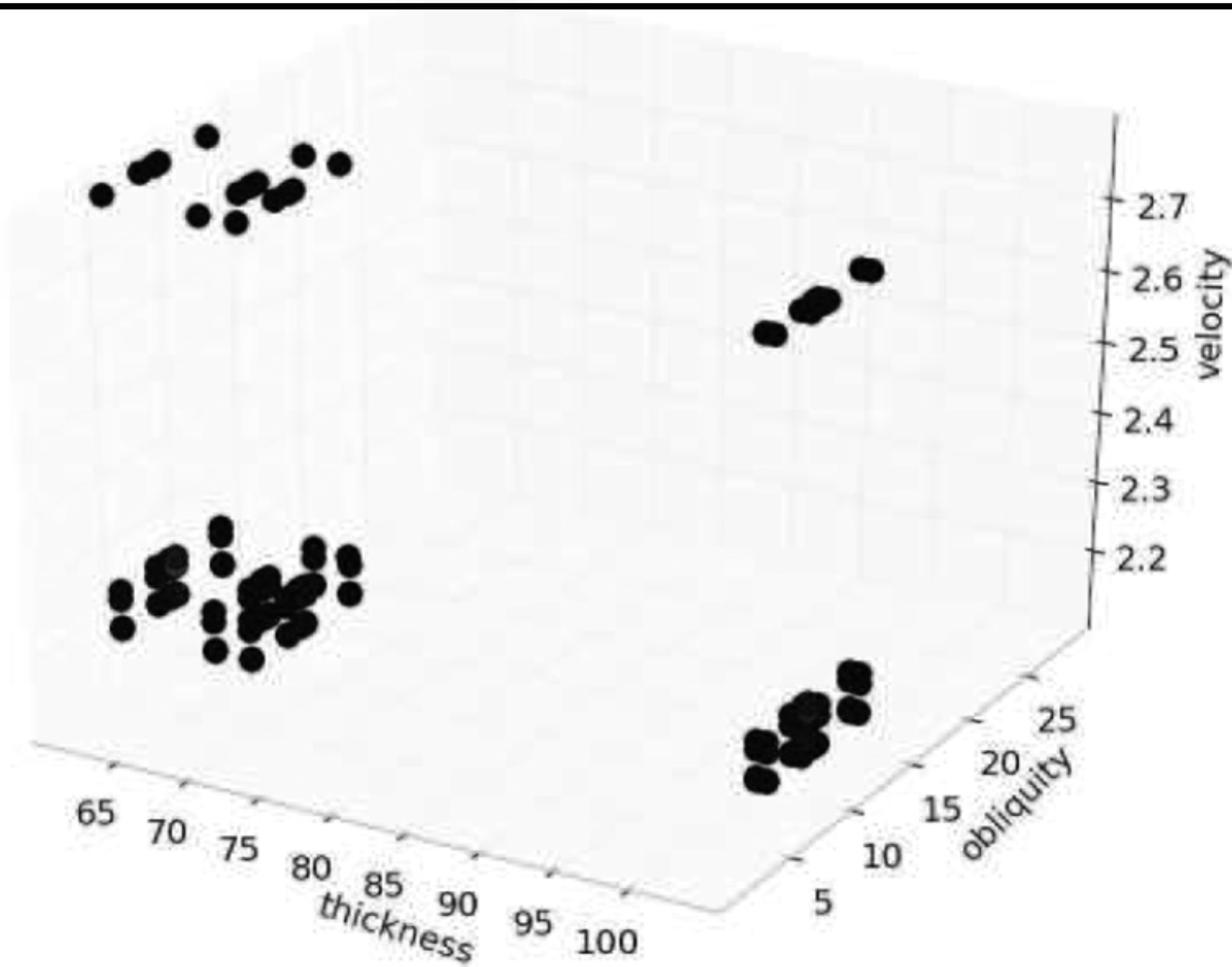
- OUQ is an optimization problem to find the rigorous bounds on system behavior
  - all information is captured as constraints
  - constraints restrict the set of all possible solutions (by directly constraining solution space)
  - systems with minimal to no experimental data or unobserved rare events that govern system behavior
- instead of selecting a "best" model or distribution or prior, we can optimize over all possible models, distributions, or priors.
  - selecting a model or distribution is treated as an assumption or information (i.e. a constraint)
  - our "prior" step becomes one of quantifying all the knowledge we have about the problem, and then encoding that knowledge as constraints

initial basis for a probability distribution…



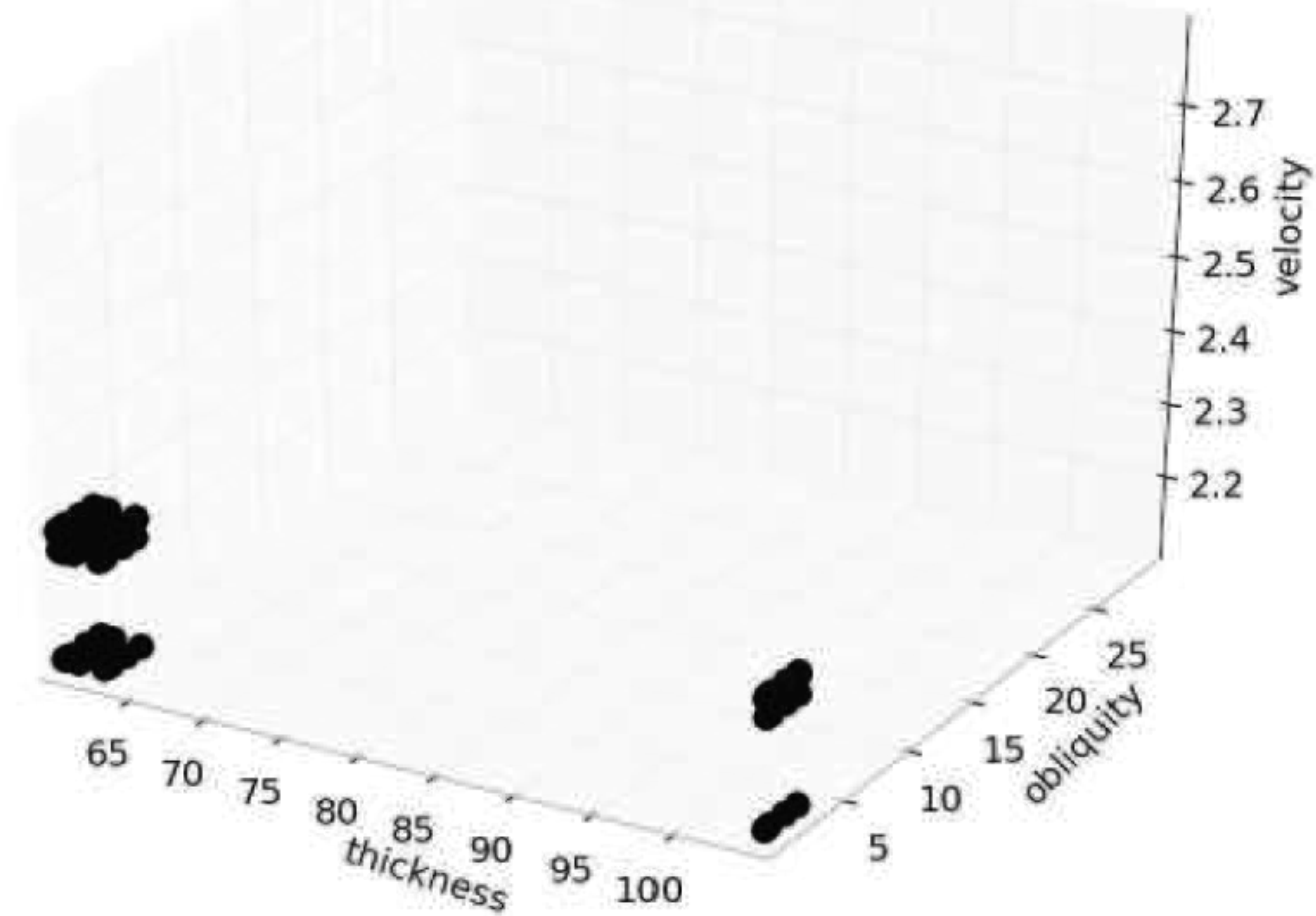
Support Points at iteration 0

…solver looks for extremal cases……



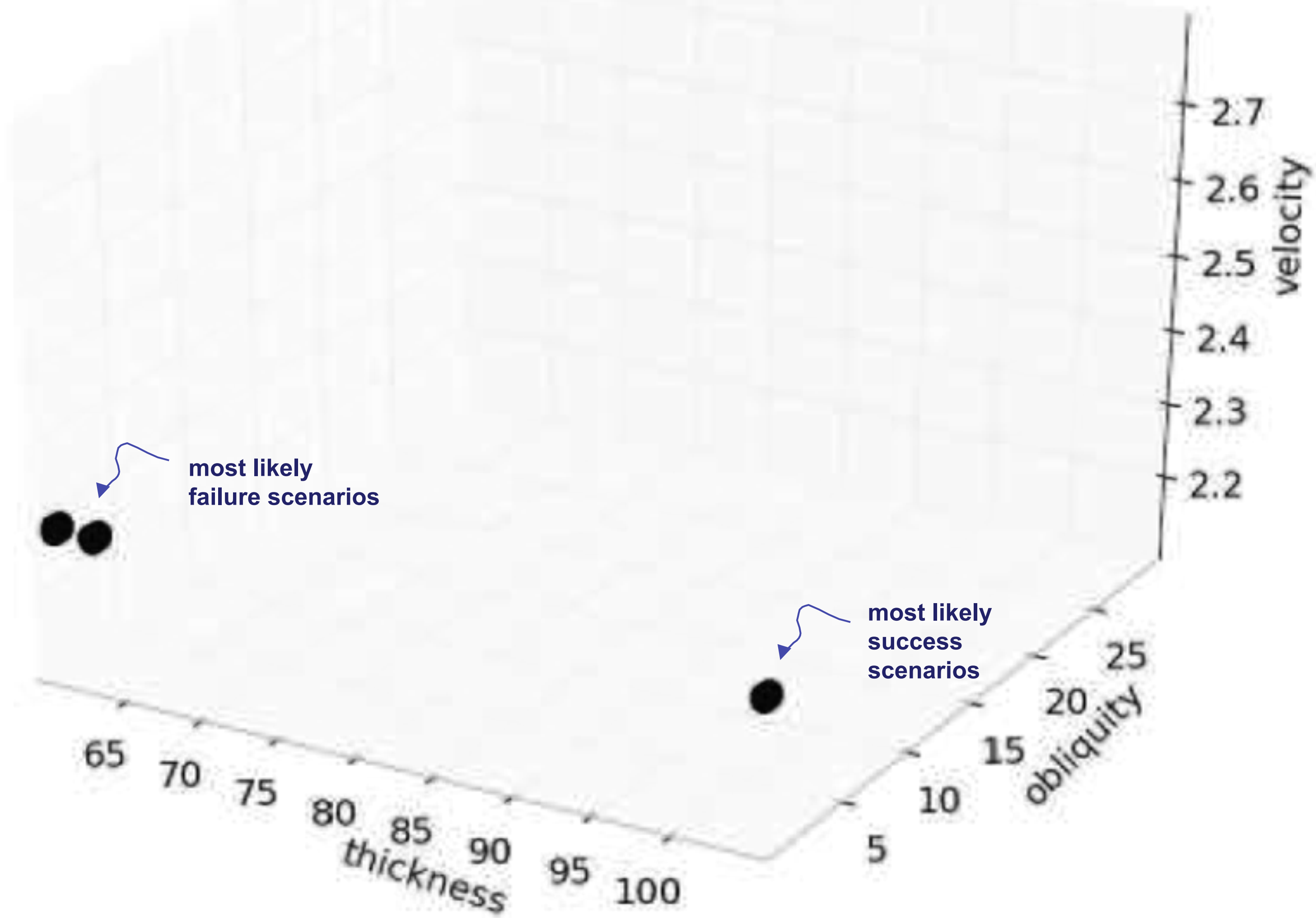
Support Points at iteration 1000

…collapses candidate scenarios…



Support Points at iteration 3000

…and solves for probability of failure



Support Points at iteration 7100

OUQ: a robust unifying UQ formulation

$$\mathcal{A} := \left\{ (g, \mu) \middle| \begin{array}{c} (g \colon \mathcal{X} \to \mathbb{R}, \mu \in \mathcal{P}(\mathcal{X})) \text{ is consistent with} \\ \text{all given information about the real system } (G, \mathbb{P}) \\ (e.g. \text{ legacy data, first principles, expert judgement}) \end{array} \right\}.$$

- Optimal bounds on the quantity of interest $\mathbb{E}_{X \sim \mathbb{P}}[q(X, G(X))]$ (optimal w.r.t. the information encoded in $\mathcal{A}$) are found by minimizing/maximizing $\mathbb{E}_{X \sim \mu}[q(X, g(X))]$ over all admissible scenarios $(g, \mu) \in \mathcal{A}$:

$$\mathcal{L}(\mathcal{A}) \leq \mathbb{E}_{X \sim \mathbb{P}}[q(X, G(X))] \leq \mathcal{U}(\mathcal{A}),$$

where $\mathcal{L}(\mathcal{A})$ and $\mathcal{U}(\mathcal{A})$ are defined by the minimization and maximization problems

extremes are bound by information in the form of constraints

formulated to handle UQ for catastrophic rare-events

$$\mathcal{L}(\mathcal{A}) := \inf_{(g,\mu) \in \mathcal{A}} \mathbb{E}_{X \sim \mu}[q(X, g(X))],$$

$$\mathcal{U}(\mathcal{A}) := \sup_{(g,\mu) \in \mathcal{A}} \mathbb{E}_{X \sim \mu}[q(X, g(X))].$$

the math is simple, but infinite dimensional

We formulate statistical quantities as optimizations, where ***x*** are physical values, **λ** are constants that are model-dependent, ***μ*** is a probability distribution, and ***A*** is all the information we have about the system.

model error $|F(x) - F'(x,\lambda)|$

statistical error $\mathbb{E}\left[|F(x) - F'(x,\lambda)|^2\right]$

model uncertainty $\mathbb{P}\left[|F(x) - F'(x,\lambda)| \geq a\right]$

likelihood $\mathbb{P}\left[|F(x) - F'(x,\lambda)| \geq a\right] \leq \epsilon$

bound on statistical error $\sup_{\mu \in \mathcal{A}} \mathbb{E}_\mu\left[|F(x) - F'(x,\lambda)|^2\right]$

optimal statistical estimator $\inf_{F'} \sup_{\mu \in \mathcal{A}} \mathbb{E}_\mu\left[|F(x) - F'(x,\lambda)|^2\right]$

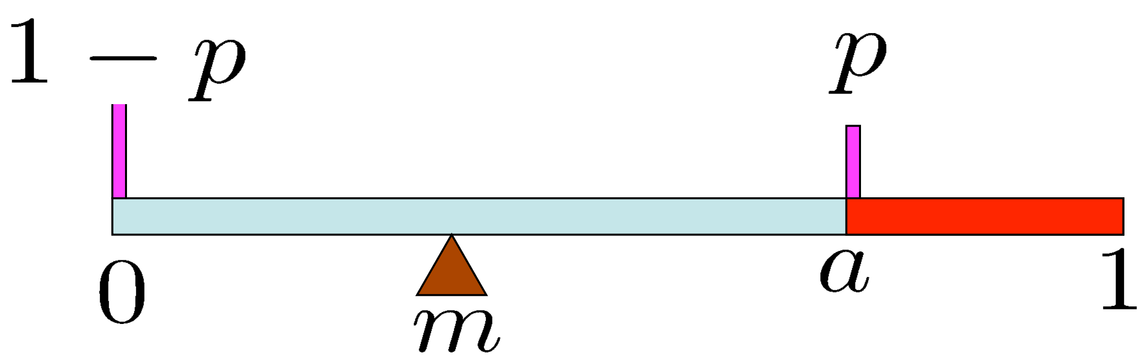
for admissible scenario (g, ν) for the unknown reality $(G, \mathbb{P})$

enables rigorous calculation of bounds on system behavior

note the optimal model is the most robust (e.g. the bounds minimally change on system change)

a simple infinite dimensional problem

You are given one pound of playdoh.
How much mass can you put above a while keeping the seesaw balanced around m?



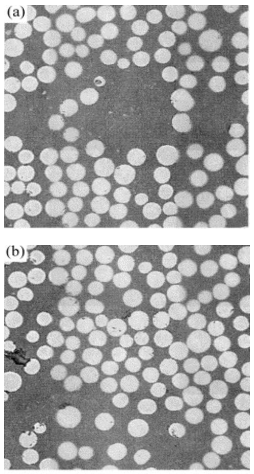
the answer $\begin{cases} \max p \\ \text{subject to } a\,p \le m \end{cases}$

the question

$$\sup_{\mu \in \mathcal{A}} \mu\big[X \ge a\big] = \frac{m}{a}$$

$$\mathcal{A} = \{\mu \in \mathcal{M}([0,1]) \,|\, \mathbb{E}_\mu[X] \le m\}$$

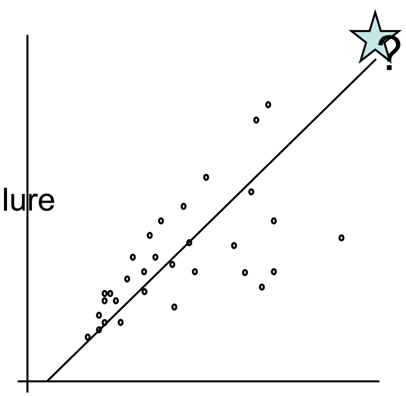
quantification in microstructure modeling



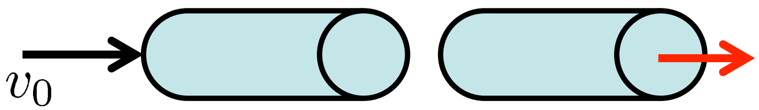
real system → QOI (quantity of interest)

model → predicted QOI

- motivating questions:
  - How "good" is my model?
  - How can I best improve my model?
  - Given the uncertainty on microstructure does it make sense to perform an expensive simulation?
  - Is there a "best" representation of the microstructure? How can I find it?
  - Can I turn the problem of finding the best model given computational constraints and available information into an algorithm?

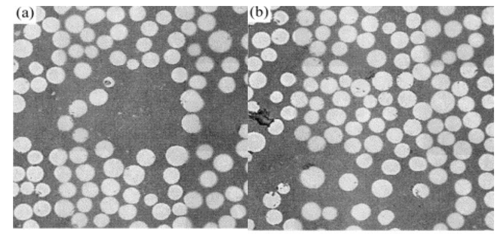


target: UQ for shock in microstructures

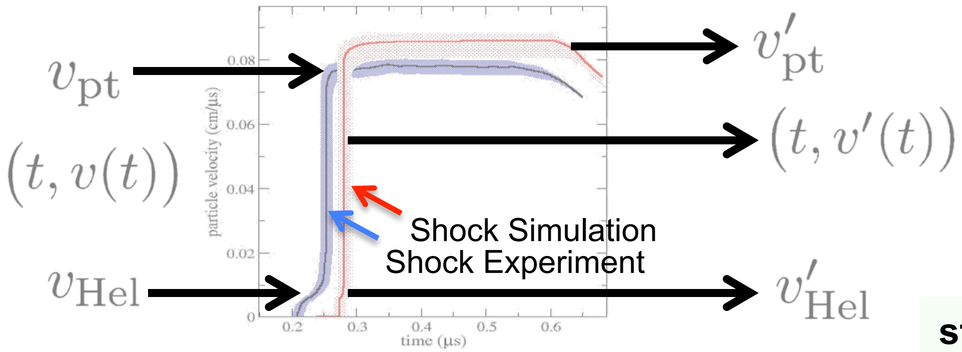


$(t, v(t))$ curve $\quad (t, v'(t))$ curve

strength models in shock waves



we want to estimate **we can compute**



model error $\left|v_{\text{Hel}}(x) - v'_{\text{Hel}}(x,\lambda)\right|$

statistical error $\mathbb{E}\left[\left|v_{\text{Hel}}(x) - v'_{\text{Hel}}(x,\lambda)\right|^2\right]$

model uncertainty $\mathbb{P}\left[\left|v_{\text{Hel}}(x) - v'_{\text{Hel}}(x,\lambda)\right| \geq a\right]$

failure $\mathbb{P}\left[\left|v_{\text{Hel}}(x) - v'_{\text{Hel}}(x,\lambda)\right| \geq a\right] \leq \varepsilon$

We have incomplete information on the distribution of x

We know v_0, h, r only up to some tolerance

$$h \in [h_{\min}, h_{\max}],\ \mathbb{E}[h] = m,\ \mathrm{Var}(h) \leq \sigma$$

We have incomplete information on the distribution of microstructure and chemical composition

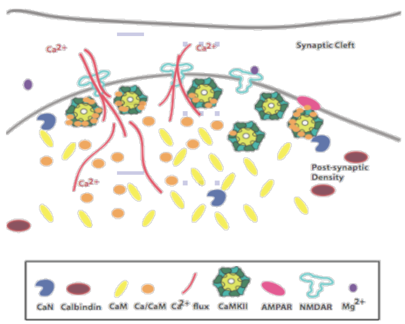
Volume fractions of iron, carbon, ...

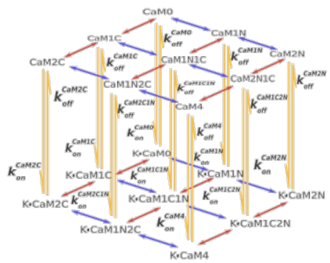
Average grain orientation and size,correlation between grain orientations as a function of distance, ...

optimal bound on the statistical error $\sup_{\mu\in\mathcal{A}} \mathbb{E}_\mu\left[\left|v_{\text{Hel}}(x) - v'_{\text{Hel}}(x,\lambda)\right|^2\right]$

used for model-parameter sensitivity

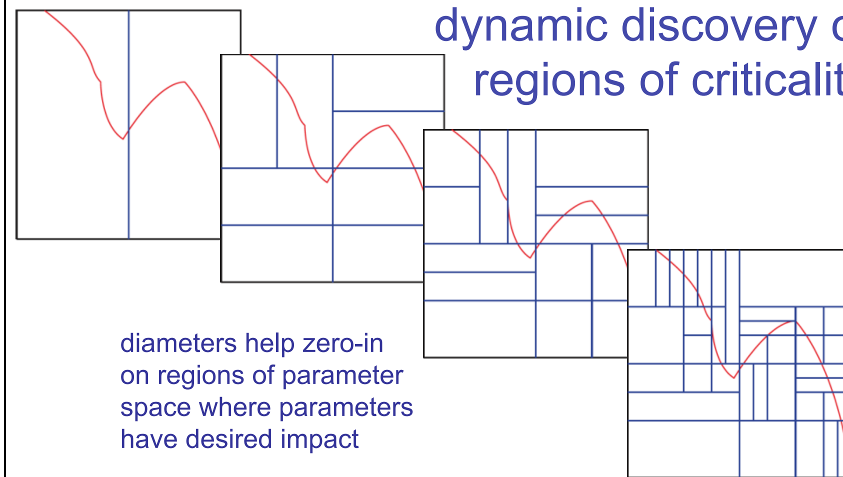
sensitivity in synaptic reaction networks





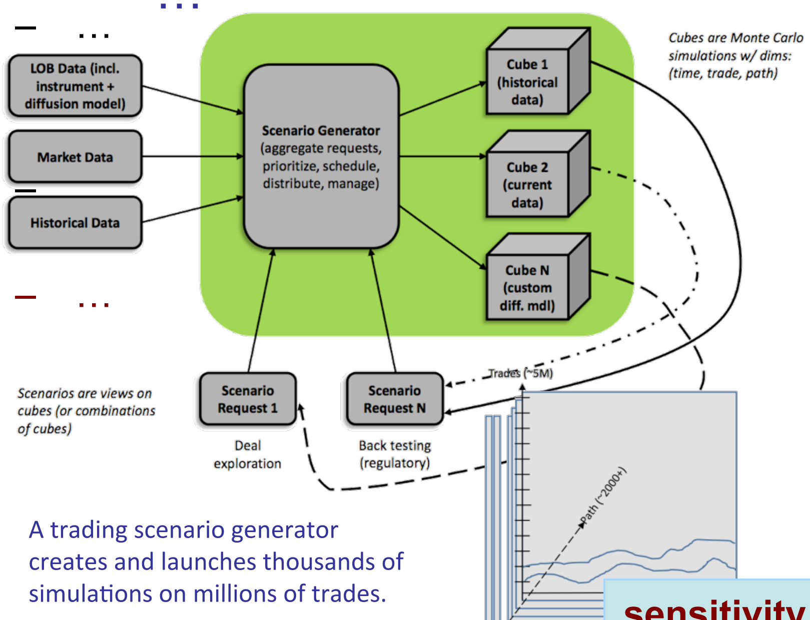
left: schematic of major players in the synaptic reaction network

dynamic discovery of regions of criticality



diameters help zero-in on regions of parameter space where parameters have desired impact

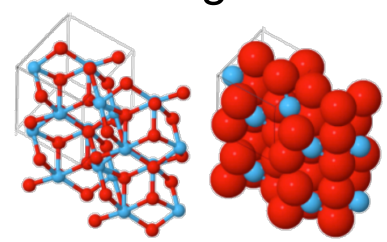
large-scale calculations of risk



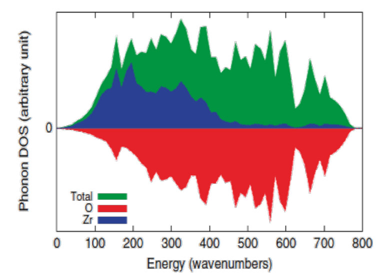
A trading scenario generator creates and launches thousands of simulations on millions of trades.

$$\text{sensitivity} = -\,|F(x') - F(x)|^2$$

sensitivity of thermodynamic peak-broadening to bond anharmonicity



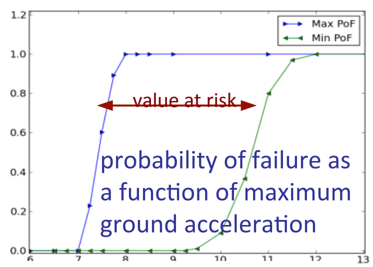
Crystal structure of monoclinic zirconia, with oxygen in red and zirconium in blue.



The partial density of states at 295 K calculated by GULP shows Zr dominates the lower energy modes.

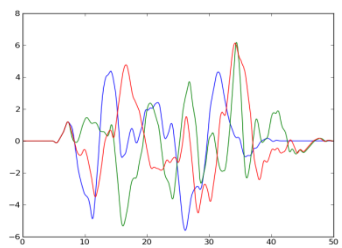
used for probability of system failure

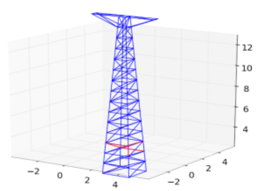
probability of elastoplastic failure under strain due to ground acceleration



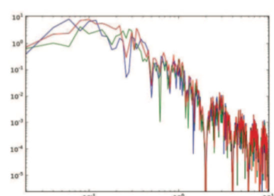
probability of failure as a function of maximum ground acceleration

typical scenarios for resulting ground acceleration



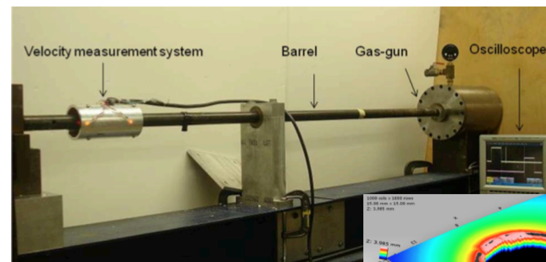


a truss structure

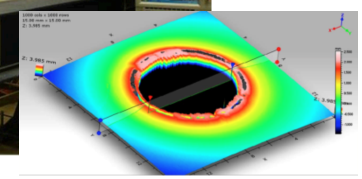


when axial strain occurs near truss resonance modes, failure can occur

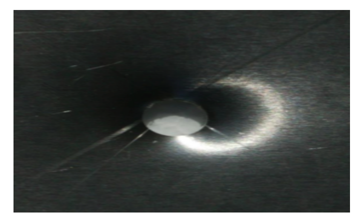
UQ for solid mechanics of hypervelocity ballistic impact

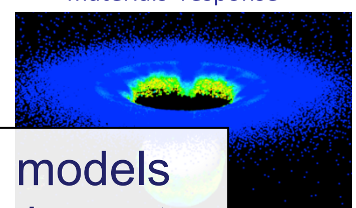


hypervelocity ballistics launcher and measurement system



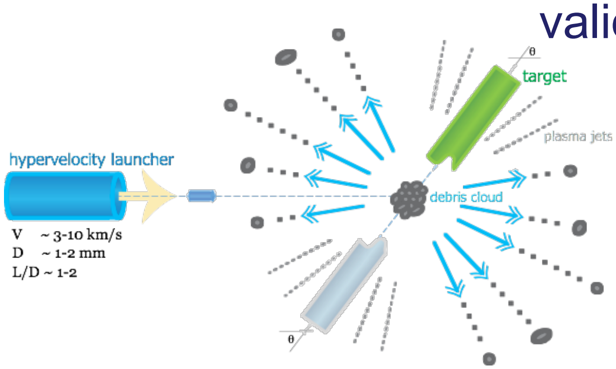
an impact simulation is used to quickly test materials response



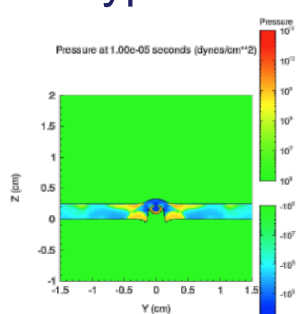


the area of the above hole is determined by a laser probe

validation of strength models for hypervelocity impact



Schematic of hypervelocity impact for a spherical steel projectile fired at a stainless steel plate.



A Von Mises yield strength model with velocity = 100 m/s is shown 5 s after impact.

A feasible set defined by bounds

$(h,\theta,v) \in [1.52, 2.67]$ mm

$\times [0, \frac{\pi}{6}] \times [2.1, 2.8]$ km/s

...and a mean constraint on area

$\mathbb{E}[H(h,\theta,v)] \in [5.5, 7.5]\ \mathrm{mm}^2$

enables the formulation of better models

We can hypothesize measurements of new information (say, a new constraint on the median of velocity, or on the angle of impact), and then optimize to see how that new information would alter the probability of the critical event.

Admissible scenarios, $\mathcal{A}$	$\mathcal{U}(\mathcal{A})$	Method
$\mathcal{A}_{\mathsf{McD}}$: independence, oscillation and mean constraints (exact response H not given)	$\leq 66.4\%$ $= 43.7\%$	McD. ineq. Opt. McD.
$\mathcal{A} := \{(f, \mu) \mid f = H \text{ and } \mathbb{E}_\mu[H] \in [5.5, 7.5]\}$	$\overset{\text{num}}{=} 37.9\%$	OUQ
$\mathcal{A} \cap \left\{(f, \mu) \,\middle|\, \begin{array}{c} \mu\text{-median velocity} \\ = 2.45\,\mathsf{km} \cdot \mathsf{s}^{-1} \end{array}\right\}$	$\overset{\text{num}}{=} 30.0\%$	OUQ
$\mathcal{A} \cap \left\{(f, \mu) \,\middle|\, \mu\text{-median obliquity} = \frac{\pi}{12}\right\}$	$\overset{\text{num}}{=} 36.5\%$	OUQ
$\mathcal{A} \cap \left\{(f, \mu) \,\middle|\, \text{obliquity} = \frac{\pi}{6}\ \mu\text{-a.s.}\right\}$	$\overset{\text{num}}{=} 28.0\%$	OUQ

We keep trying to **design possible "experiments"** to find the information set that certifies the system as "safe" (not failing within the given tolerance)

more questions == new objective functions

- Can I use OUQ to find if I can use sampling statistics?

- Can I find a suitable reduced-dimensional model?

- Which data points are the “most impactful”?
  - after 1 year of hypervelocity impact experiments, post-analysis found that only 2 of the nearly 50 shots impacted the probability of failure bounds
  - better: use the statistics as an guide for where to shoot next

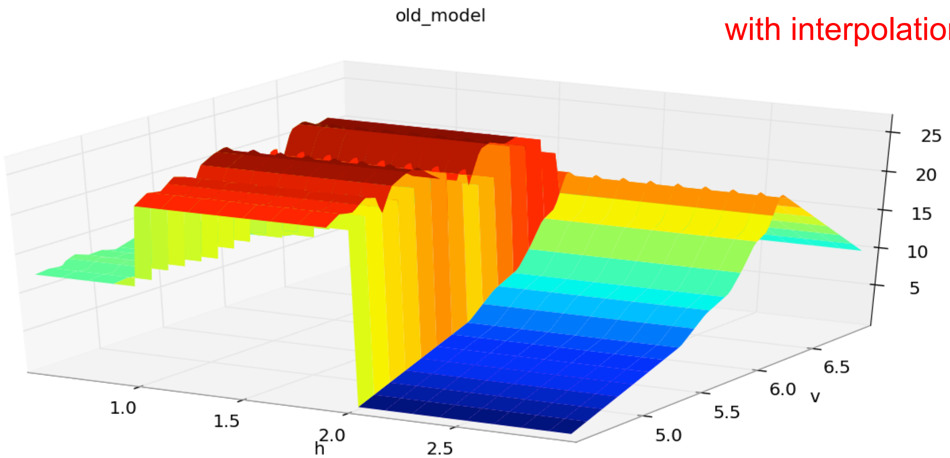
- Formulation of these problems as OUQ questions, under the mystic framework, is designed to run asynchronously and to be resilient to failure.
  - in many cases, this requires depth 5 optimization problems

when bounds are violated, look for bugs!

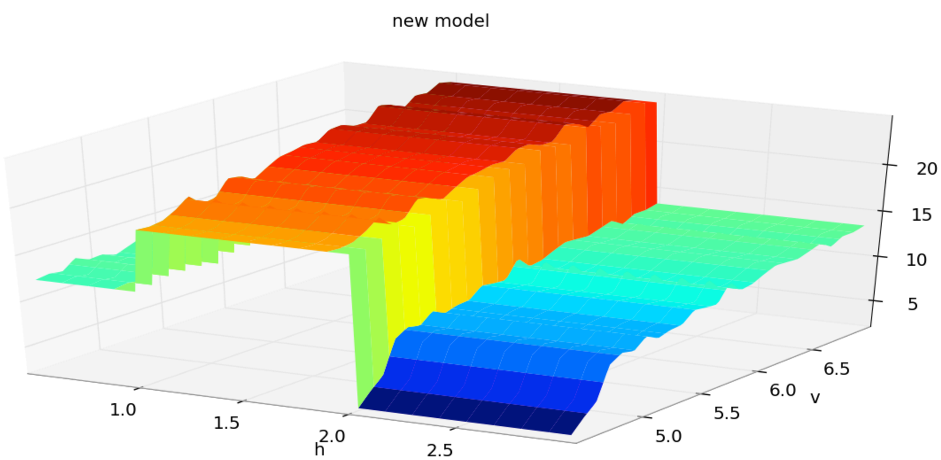
in an OUQ calculation of probability of failure, the results began violating the calculated system bounds

subsequent OUQ analysis on elements of the calculation discovered a version update to code for kriging interpolation came with a new bug

since this error represented a violation of our assumptions (information) about the problem, it led to results that violated the "worst case" bounds.



with interpolation bug



without interpolation bug

references

[1] http://arxiv.org/abs/1308.6306

[2] http://arxiv.org/abs/1304.6772

[3] **SIAM Rev 2013** http://arxiv.org/abs/1009.0679

[4] **M2AN 2013** http://arxiv.org/abs/1202.1928

[5] http://pythonhosted.org/mystic

[6] https://github.com/uqfoundation

This is not in any way a solved problem, and I believe is just the opening gambit.

End Presentation